



iceMASTER™ Software Version 3.3 Supplement

Module:TA Source File:takf511.asm

```
90     /* scan through nodes in list */
91     p_list = list_head;
92     while (p_list != NULL) {
93         ij_uival = p_list->list_uival;
94         p_list = p_list->list_p_nxt;
95     } /*while p_list*/
96
97     iteration += 1.1;
98     main_loop_factor++;
99
100 End_Loop: ; /* (for demo's -- in case lines inserted/deleted) */
101
102 } /*while TRUE*/
103
104 } /*main*/
```

Enter:Change Watch Ctrl-B:Browse

Addr	Addr	Data Type	Value	Ctrl-B:Browse
X:009C	X:0097	X:0092 struct List	struct List	11
X:009C	X:0097	X:0092 struct List	list_uival 3	13
X:009C	X:0097	X:0092 uint	list_p_nxt (null)	15
X:008C	X:0094	X:0092 struct List	list_p_nxt (null)	17
X:008D	Enter:Change	X:0094 ->struct List	list_p_nxt (null)	19
X:00A8	->struct List	p_list	(null)	21
C:0000?struct List	*p_list		struct List	
C:0000?uint			list_uival 517	
C:0002?->struct List			list_p_nxt ?:FFFFCCEFFF	
X:0092	[3]struct List	nodes	[3]struct List	
X:0092	struct List		[0] struct List	
X:0092	uint		list_uival 3	
X:0094	->struct List		list_p_nxt (null)	
X:0097	struct List		[1] struct List	
X:0097	uint		list_uival 7	
X:0099	->struct List		list_p_nxt X:0092	
X:009C	struct List		[2] struct List	
X:009C	uint		list_uival 11	
X:009E	->struct List		list_p_nxt X:0097	
X:00A5	X:0092 struct List	list_head	X:009C	
X:0080	float	iteration	2.2	

Tab/Shift-Tab:Next/Prev window (Keypad):Scroll Esc:Exit

© 1993 by MetaLink Corporation

All rights are reserved.

This manual may not, in whole or part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without the prior agreement and written permission of MetaLink Corporation.

ICE-HOUSE®, MetaICE-CHEST®, MetaICE®, MicroICE®, and MicroICE® +, and the MetaLink logo are registered trademarks of MetaLink Corporation.

iceMASTER-PE™, iceMASTER™, ICELink™, MetaLink™, and the combination of MetaICE®, iceMASTER-PE™, iceMASTER™, MicroICE®, or MicroICE® + and an alphabetic or numeric suffix are claimed trademarks of MetaLink Corporation and may only be used to describe MetaLink products.

National Semiconductor® and National Semiconductor Corp.® are registered trademarks of National Semiconductor Corporation.

MOLE™ is a trademark of National Semiconductor Corporation.

IBM® is a registered trademark of IBM Corporation.

PC-DOS is a trademark of IBM Corporation.

MS-DOS® is a registered trademark of Microsoft Corporation.

Windows™ is a trademark of Microsoft Corporation.

MCS® and Intel® are registered trademarks of Intel Corporation.

Archimedes™ is a trademark of Archimedes Software, Inc.

Franklin™ is a trademark of Franklin Software, Inc.

MetaLink Corporation reserves the right to make improvements in the products described in this manual as well as the manual itself at any time and without notice.

Table of Contents

Chapter 1: Introduction

1-1

Chapter 2: Tutorial

2-1

Program TA.C	2-2
Watch Window	2-4
Browsing	2-9
Display/Alter Expression Command	2-12
Source/Symbols Commands	2-16
Improved Support for "Unsupported" Programs	2-19

Chapter 3: Reference

3-1

Expression Input	3-2
Identifiers	3-2
Constants	3-2
Integer Constants	3-2
Explicit Radix	3-2
No	3-4
Explicit Radix	3-4
Floating-Point Constants	3-5
Character Constants	3-5
String Constants	3-6
Expressions	3-6
Primary	3-6
Postfix	3-8
Unary	3-8
Cast	3-8
Multiplicative	3-9
Additive	3-9
Display Formats	3-10
Addresses	3-10
Type Names	3-11
Fundamental Types	3-11

Derived/Aggregate Types	3-12
Values	3-12
Fundamental Types	3-12
Pointers	3-13
Generic Pointers	3-13
Memory-Specific Pointers	3-13
NULL Pointers	3-13
Structures, Unions and Arrays	3-14
Functions	3-14
Configure Options Expressions Command	3-15
Configure Options Expressions Radix	3-15
Configure Options Expressions Unknown data type size	3-16
Configure Options Expressions Browse Windows Stack	3-16
Configure Options Expressions Struct/Union Indentation	3-17
Configure Options Expressions Member Names	3-18
Configure Options Expressions Array Subscript Values	3-19
Configure Options Expressions unknown/untyped	
Configure Options Expressions short/ushort	
Configure Options Expressions int/uint	
Configure Options Expressions long/ulong	
Configure Options Expressions float	
Configure Options Expressions double	3-20
Configure Options Expressions Command Combinations	3-21
Watch Window	3-22
Commands	3-22
Display Format	3-22
Browsing	3-23
Source/Symbols Displays	3-24
Display/Alter Expression	3-25
Change Box	3-26
Browse Windows	3-27

Compilers	3-29
Keil/Franklin C51	3-29
Recommended Compilation Options	3-29
Recommended Linking Options	3-29
Pointers	3-29
NULL	3-29
'typedef' Names	3-30
Enumeration Types	3-30
'double' Types	3-30
Batch "Make" Files	3-30
IAR/Archimedes C-51	3-31
Recommended Compilation Options	3-31
Recommended Linking Options	3-31
Pointers	3-31
NULL	3-31
Reading	3-31
Writing	3-31
'typedef' Names	3-32
Enumeration Types	3-32
'double' Types	3-32
Batch "Make" Files	3-32

Appendix A: Source Window & Assembler/Disassembler A-1

Source Window	A-1
Source Window Control	A-1
Browse	A-1
Change Module	A-2
Mode	A-2
Mode Code	A-2
Mode Mixed	A-2
Mode HLL	A-2
Label-Synch	A-2
Set Break	A-2

<i>Set Break Toggle</i>	A-2
<i>Set Break Run Until</i>	A-3
<i>Assemble</i>	A-3
<i>Display/Alter Asm/Dasm</i>	A-3
<i>Display/Alter Asm/Dasm Browse</i>	A-3
<i>Display/Alter Asm/Dasm Assemble</i>	A-5
<i>Display/Alter Asm/Dasm Mode</i>	A-5
<i>Display/Alter Asm/Dasm Label-synch</i>	A-5
<i>Display/Alter Asm/Dasm Write</i>	A-6

Appendix B: Virtual Memory Support

B-1

<i>Command Line Options</i>	B-1
<i>Misc Virtual Memory Command</i>	B-3

B-A

Appendix A: Source Window & Assembler/Disassembler

I-A

Source Window

I-A

Source Window Control

I-A

Source

S-A

Source Window

S-A

Notes

S-A

Source|B-27, J-10, S-3

S-A

Notes|C-8

S-A

Notes|H-17

S-A

Notes|I-15

S-A

Notes|J-10

Browsing Methods

Chapter 1: Introduction

iceMASTER Host Software Version 3.3 contains these major enhancements:

- All address, length or count specifications can now be general expressions, rather than just a single symbol or constant.
- Support for long integer, single-precision (**float**) and double-precision (**double**) floating-point variables
 - Display
 - Input
- Full support for structures, unions, arrays and pointers
 - Display/Change
 - * Watch Window
 - * *Display/Alter|Expression* command
 - * *Source/Symbols|Global* command
 - * *Source/Symbols|Local* command
 - * *Source/Symbols|Address* command
 - * *Source/Symbols|Alpha* command
 - Full structure, union, array & pointer support for these compilers:
 - * Keil/Franklin C51
 - * IAR/Archimedes C-51
 - * (others coming soon)
- Data Structure Browser/Inspector
 - Inspect structures, unions and arrays
 - Follow pointers
 - Change any value at any point along the way
 - Backtrack
- True Expressions in Watch Window (not just static addresses)
 - Indirection through NULL or invalid pointers is flagged in the display.
- OS-Escape ("Shell-Out") to DOS during Emulation
 - OS-Escape Hot Key: **Alt-O**
 - Emulation continues while working in DOS.
 - All DOS memory, except about 7K-10K bytes, available during all OS-Escapes
- Supports all *iceMASTER* hardware products
 - *iceMASTER* Model 200/400 Emulators (MCS-51, COP8, 68HC11 & 68HC05)
 - *iceMASTER-PE* Emulators
 - *iceMASTER-COP8* Debug Modules

The following features were added in earlier versions of the *iceMASTER* host software. Depending on which *iceMASTER* hardware product you purchased, these features may or may not be documented in the primary manual:

- Source Window & Assembler/Disassembler Enhancements
(software version 3.2 – see Appendix A)
 - "Source-Only" display mode option in Source Window
 - Assembler/Disassembler Window fully-scrollable
 - Set permanent break-points directly in Source or Assembler/Disassembler Windows
 - Set temporary break-points directly in Source Window
- Virtual Memory Support
(software version 3.1 – see Appendix B)
 - Allows host software to use Expanded Memory for large programs

Chapter 2: Tutorial

Here is short tour through some of the new features in the *iceMASTER* host software. The tutorial is based primarily on the C program TA.C shown on the following pages.

```
 1  /* 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 339 340 341 342 343 344 345 346 347 348 349 349 350 351 352 353 354 355 356 357 358 359 359 360 361 362 363 364 365 366 367 368 369 369 370 371 372 373 374 375 376 377 378 379 379 380 381 382 383 384 385 386 387 388 389 389 390 391 392 393 394 395 396 397 398 399 399 400 401 402 403 404 405 406 407 408 409 409 410 411 412 413 414 415 416 417 418 419 419 420 421 422 423 424 425 426 427 428 429 429 430 431 432 433 434 435 436 437 438 439 439 440 441 442 443 444 445 446 447 448 449 449 450 451 452 453 454 455 456 457 458 459 459 460 461 462 463 464 465 466 467 468 469 469 470 471 472 473 474 475 476 477 478 479 479 480 481 482 483 484 485 486 487 488 489 489 490 491 492 493 494 495 496 497 498 499 499 500 501 502 503 504 505 506 507 508 509 509 510 511 512 513 514 515 516 517 518 519 519 520 521 522 523 524 525 526 527 528 529 529 530 531 532 533 534 535 536 537 538 539 539 540 541 542 543 544 545 546 547 548 549 549 550 551 552 553 554 555 556 557 558 559 559 560 561 562 563 564 565 566 567 568 569 569 570 571 572 573 574 575 576 577 578 579 579 580 581 582 583 584 585 586 587 588 589 589 590 591 592 593 594 595 596 597 598 599 599 600 601 602 603 604 605 606 607 608 609 609 610 611 612 613 614 615 616 617 618 619 619 620 621 622 623 624 625 626 627 628 629 629 630 631 632 633 634 635 636 637 638 639 639 640 641 642 643 644 645 646 647 648 649 649 650 651 652 653 654 655 656 657 658 659 659 660 661 662 663 664 665 666 667 668 669 669 670 671 672 673 674 675 676 677 678 679 679 680 681 682 683 684 685 686 687 688 689 689 690 691 692 693 694 695 696 697 698 699 699 700 701 702 703 704 705 706 707 708 709 709 710 711 712 713 714 715 716 717 718 719 719 720 721 722 723 724 725 726 727 728 729 729 730 731 732 733 734 735 736 737 738 739 739 740 741 742 743 744 745 746 747 748 749 749 750 751 752 753 754 755 756 757 758 759 759 760 761 762 763 764 765 766 767 768 769 769 770 771 772 773 774 775 776 777 778 779 779 780 781 782 783 784 785 786 787 788 789 789 790 791 792 793 794 795 796 797 798 799 799 800 801 802 803 804 805 806 807 808 809 809 810 811 812 813 814 815 816 817 818 819 819 820 821 822 823 824 825 826 827 828 829 829 830 831 832 833 834 835 836 837 838 839 839 840 841 842 843 844 845 846 847 848 849 849 850 851 852 853 854 855 856 857 858 859 859 860 861 862 863 864 865 866 867 868 869 869 870 871 872 873 874 875 876 877 878 879 879 880 881 882 883 884 885 886 887 888 889 889 890 891 892 893 894 895 896 897 898 899 899 900 901 902 903 904 905 906 907 908 909 909 910 911 912 913 914 915 916 917 918 919 919 920 921 922 923 924 925 926 927 928 929 929 930 931 932 933 934 935 936 937 938 939 939 940 941 942 943 944 945 946 947 948 949 949 950 951 952 953 954 955 956 957 958 959 959 960 961 962 963 964 965 966 967 968 969 969 970 971 972 973 974 975 976 977 978 979 979 980 981 982 983 984 985 986 987 988 989 989 990 991 992 993 994 995 996 997 998 999 999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1029 1030 1031 1032 1033 1034 1035 1036 1037 1038 1039 1039 1040 1041 1042 1043 1044 1045 1046 1047 1048 1049 1049 1050 1051 1052 1053 1054 1055 1056 1057 1058 1059 1059 1060 1061 1062 1063 1064 1065 1066 1067 1068 1069 1069 1070 1071 1072 1073 1074 1075 1076 1077 1078 1079 1079 1080 1081 1082 1083 1084 1085 1086 1087 1088 1089 1089 1090 1091 1092 1093 1094 1095 1096 1097 1098 1099 1099 1100 1101 1102 1103 1104 1105 1106 1107 1108 1109 1109 1110 1111 1112 1113 1114 1115 1116 1117 1118 1119 1119 1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139 1139 1140 1141 1142 1143 1144 1145 1146 1147 1148 1149 1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1159 1159 1160 1161 1162 1163 1164 1165 1166 1167 1168 1169 1169 1170 1171 1172 1173 1174 1175 1176 1177 1178 1179 1179 1180 1181 1182 1183 1184 1185 1186 1187 1188 1189 1189 1190 1191 1192 1193 1194 1195 1196 1197 1198 1198 1199 1199 1200 1201 1202 1203 1204 1205 1206 1207 1208 1209 1209 1210 1211 1212 1213 1214 1215 1216 1217 1218 1219 1219 1220 1221 1222 1223 1224 1225 1226 1227 1228 1229 1229 1230 1231 1232 1233 1234 1235 1236 1237 1238 1239 1239 1240 1241 1242 1243 1244 1245 1246 1247 1248 1249 1249 1250 1251 1252 1253 1254 1255 1256 1257 1258 1259 1259 1260 1261 1262 1263 1264 1265 1266 1267 1268 1269 1269 1270 1271 1272 1273 1274 1275 1276 1277 1278 1279 1279 1280 1281 1282 1283 1284 1285 1286 1287 1288 1289 1289 1290 1291 1292 1293 1294 1295 1296 1297 1298 1298 1299 1299 1300 1301 1302 1303 1304 1305 1306 1307 1308 1309 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1319 1319 1320 1321 1322 1323 1324 1325 1326 1327 1328 1329 1329 1330 1331 1332 1333 1334 1335 1336 1337 1338 1339 1339 1340 1341 1342 1343 1344 1345 1346 1347 1348 1349 1349 1350 1351 1352 1353 1354 1355 1356 1357 1358 1359 1359 1360 1361 1362 1363 1364 1365 1366 1367 1368 1369 1369 1370 1371 1372 1373 1374 1375 1376 1377 1378 1379 1379 1380 1381 1382 1383 1384 1385 1386 1387 1388 1389 1389 1390 1391 1392 1393 1394 1395 1396 1397 1398 1398 1399 1399 1400 1401 1402 1403 1404 1405 1406 1407 1408 1409 1409 1410 1411 1412 1413 1414 1415 1416 1417 1418 1419 1419 1420 1421 1422 1423 1424 1425 1426 1427 1428 1429 1429 1430 1431 1432 1433 1434 1435 1436 1437 1438 1439 1439 1440 1441 1442 1443 1444 1445 1446 1447 1448 1449 1449 1450 1451 1452 1453 1454 1455 1456 1457 1458 1459 1459 1460 1461 1462 1463 1464 1465 1466 1467 1468 1469 1469 1470 1471 1472 1473 1474 1475 1476 1477 1478 1479 1479 1480 1481 1482 1483 1484 1485 1486 1487 1488 1489 1489 1490 1491 1492 1493 1494 1495 1496 1497 1498 1498 1499 1499 1500 1501 1502 1503 1504 1505 1506 1507 1508 1509 1509 1510 1511 1512 1513 1514 1515 1516 1517 1518 1519 1519 1520 1521 1522 1523 1524 1525 1526 1527 1528 1529 1529 1530 1531 1532 1533 1534 1535 1536 1537 1538 1539 1539 1540 1541 1542 1543 1544 1545 1546 1547 1548 1549 1549 1550 1551 1552 1553 1554 1555 1556 1557 1558 1559 1559 1560 1561 1562 1563 1564 1565 1566 1567 1568 1569 1569 1570 1571 1572 1573 1574 1575 1576 1577 1578 1579 1579 1580 1581 1582 1583 1584 1585 1586 1587 1588 1589 1589 1590 1591 1592 1593 1594 1595 1596 1597 1598 1598 1599 1599 1600 1601 1602 1603 1604 1605 1606 1607 1608 1609 1609 1610 1611 1612 1613 1614 1615 1616 1617 1618 1619 1619 1620 1621 1622 1623 1624 1625 1626 1627 1628 1629 1629 1630 1631 1632 1633 1634 1635 1636 1637 1638 1639 1639 1640 1641 1642 1643 1644 1645 1646 1647 1648 1649 1649 1650 1651 1652 1653 1654 1655 1656 1657 1658 1659 1659 1660 1661 1662 1663 1664 1665 1666 1667 1668 1669 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1679 1680 1681 1682 1683 1684 1685 1686 1687 1688 1689 1689 1690 1691 1692 1693 1694 1695 1696 1697 1698 1698 1699 1699 1700 1701 1702 1703 1704 1705 1706 1707 1708 1709 1709 1710 1711 1712 1713 1714 1715 1716 1717 1718 1719 1719 1720 1721 1722 1723 1724 1725 1726 1727 1728 1729 1729 1730 1731 1732 1733 1734 1735 1736 1737 1738 1739 1739 1740 1741 1742 1743 1744 1745 1746 1747 1748 1749 1749 1750 1751 1752 1753 1754 1755 1756 1757 1758 1759 1759 1760 1761 1762 1763 1764 1765 1766 1767 1768 1769 1769 1770 1771 1772 1773 1774 1775 1776 1777 1778 1779 1779 1780 1781 1782 1783 1784 1785 1786 1787 1788 1789 1789 1790 1791 1792 1793 1794 1795 1796 1797 1798 1798 1799 1799 1800 1801 1802 1803 1804 1805 1806 1807 1808 1809 1809 1810 1811 1812 1813 1814 1815 1816 1817 1818 1819 1819 1820 1821 1822 1823 1824 1825 1826 1827 1828 1829 1829 1830 1831 1832 1833 1834 1835 1836 1837 1838 1839 1839 1840 1841 1842 1843 1844 1845 1846 1847 1848 1849 1849 1850 1851 1852 1853 1854 1855 1856 1857 1858 1859 1859 1860 1861 1862 1863 1864 1865 1866 1867 1868 1869 1869 1870 1871 1872 1873 1874 1875 1876 1877 1878 1879 1879 1880 1881 1882 1883 1884 1885 1886 1887 1888 1889 1889 1890 1891 1892 1893 1894 1895 1896 1897 1898 1898 1899 1899 1900 1901 1902 1903 1904 1905 1906 1907 1908 1909 1909 1910 1911 1912 1913 1914 1915 1916 1917 1918 1919 1919 1920 1921 1922 1923 1924 1925 1926 1927 1928 1929 1929 1930 1931 1932 1933 1934 1935 1936 1937 1938 1939 1939 1940 1941 1942 1943 1944 1945 1946 1947 1948 1949 1949 1950 1951 1952 1953 1954 1955 1956 1957 1958 1959 1959 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1998 1999 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2019 2020 2021 2022 2023 2024 2025 2026 2027 2028 2029 2029 2030 2031 2032 2033 2034 2035 2036 2037 2038 2039 2039 2040 2041 2042 2043 2044 2045 2046 2047 2048 2049 2049 2050 2051 2052 2053 2054 2055 2056 2057 2058 2059 2059 2060 2061 2062 2063 2064 2065 2066 2067 2068 2069 2069 2070 2071 2072 2073 2074 2075 2076 2077 2078 2079 2079 2080 2081 2082 2083 2084 2085 2086 2087 2088 2089 2089 2090 2091 2092 2093 2094 2095 2096 2097 2098 2098 2099 2099 2100 2101 2102 2103 2104 2105 2106 2107 2108 2109 2109 2110 2111 2112 2113 2114 2115 2116 2117 2118 2119 2119 2120 2121 2122 2123 2124 2125 2126 2127 2128 2129 2129 2130 2131 2132 2133 2134 2135 2136 2137 2138 2139 2139 2140 2141 2142 2143 2144 2145 2146 2147 2148 2149 2149 2150 2151 2152 2153 2154 2155 2156 2157 2158 2159 2159 2160 2161 2162 2163 2164 2165 2166 2167 2168 2169 2169 2170 2171 2172 2173 2174 2175 2176 2177 2178 2179 2179 2180 2181 2182 2183 2184 2185 2186 2187 2188 2189 2189 2190 2191 2192 2193 2194 2195 2196 2197 2198 2198 2199 2199 2200 2201 2202 2203 2204 2205 2206 2207 2208 2209 2209 2210 2211 2212 2213 2214 2215 2216 2217 2218 2219 2219 2220 2221 2222 2223 2224 2225 2226 2227 2228 2229 2229 2230 2231 2232 2233 2234 2235 2236 2237 2238 2239 2239 2240 2241 2242 2243 2244 2245 2246 2247 2248 2249 2249 2250 2251 2252 2253 2254 2255 2256 2257 22
```

Program T.A.C

```
1  /* ta.c */
2
3  /* #defines set on compilation command line:
4  **      K_F      Keil/Franklin
5  **      I_A      IAR/Archimedes
6  **
7  **      -----
8  **      LARGE    large memory model
9  **      COMPACT  compact memory model
10 **      SMALL    small memory model
11 **      -----
12 **      A51     8051 architecture
13 **      A11     68HC11 architecture
14 */
15 #ifdef K_F          /*{*/
16 # pragma  pl(86)      /* Listing File Page Length (lines) */
17 # pragma  pw(132)     /* Listing File Page Width  (characters) */
18 #endif /*K_F*/      /*} */
19
20 #include <stdio.h>
21 /*#include <stdarg.h> */
22 #include <string.h>
23
24 #if      defined(LARGE)      /*{{ large memory model */
25 # define  ARYSIZ 10
26 #elif    defined(COMPACT)    /*{{ compact memory model */
27 # define  ARYSIZ 7
28 #elif    defined(SMALL)     /*{{ small memory model */
29 # define  ARYSIZ 4
30 #else
31     ??? /*(force syntax error)*/
32 #endif           /*}} memory model */
33
34 # define  LIST_SIZ 3
35 # define  TRUE   1
36
37 /*****/
38 /* typedefs */
39 /*****/
40 typedef unsigned char  uchar;
41 typedef unsigned int   uint;
42
43 #if      defined(K_F)      /*{{*/
44     typedef struct List {
45         uint      list_uival;
46         struct List* list_p_nxt;
47     } LIST;
48     typedef struct List*   P_LIST;
49 #elif    defined(I_A)      /*{{*/
50     typedef struct List*   P_LIST;
51     typedef struct List {
52         uint      list_uival;
53         P_LIST   list_p_nxt;
54     } LIST;
55 #endif /*compiler*/      /*}} */

```

```

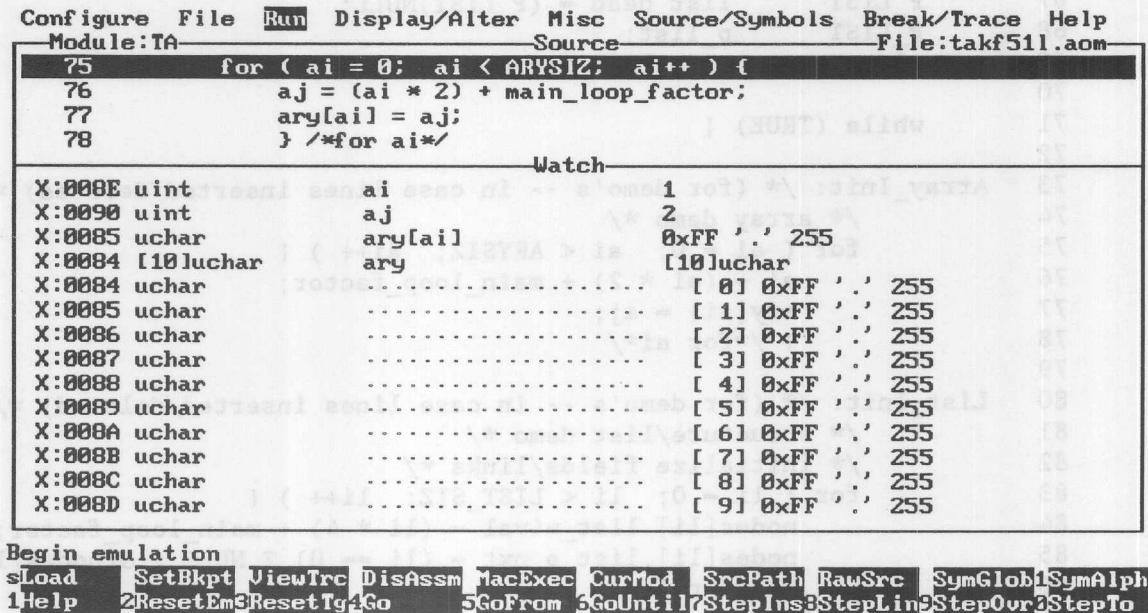
56
57                         *****/
58                         /* main() */
59                         *****/
60 main ()
61 {
62     float      iteration = 0.0;
63     uchar      ary[ARYSIZ];
64     uint       ai = 1, aj = 2;
65     LIST       nodes[LIST_SIZ];
66     uint       li, lj;
67     P_LIST     list_head = (P_LIST)NULL;
68     P_LIST     p_list;
69     uint       main_loop_factor = 1;
70
71     while (TRUE) {
72
73     Array_Init: /* (for demo's -- in case lines inserted/deleted) */
74         /* array demo */
75         for ( ai = 0; ai < ARYSIZ; ai++ ) {
76             aj = (ai * 2) + main_loop_factor;
77             ary[ai] = aj;
78         } /*for ai*/
79
80     List_Init: /* (for demo's -- in case lines inserted/deleted) */
81         /* structure/list demo */
82         /* initialize fields/links */
83         for ( li = 0; li < LIST_SIZ; li++ ) {
84             nodes[li].list_uival = (li * 4) + main_loop_factor;
85             nodes[li].list_p_nxt = (li == 0) ? NULL : &(nodes[li-1]);
86         } /*for li*/
87         list_head = &(nodes[LIST_SIZ-1]);
88
89     List_Scan: /* (for demo's -- in case lines inserted/deleted) */
90         /* scan through nodes in list */
91         p_list = list_head;
92         while (p_list != NULL) {
93             lj      = p_list->list_uival;
94             p_list = p_list->list_p_nxt;
95         } /*while p_list*/
96
97         iteration += 1.1;
98         main_loop_factor++;
99
100    End_Loop: ; /* (for demo's -- in case lines inserted/deleted) */
101
102    } /*while TRUE*/
103
104 } /*main*/

```

Watch Window

The program has two parts. The first part is a loop to initialize the elements in an array of **unsigned chars**. The second part uses an array of **structures** to implement a singly-linked list. The first sub-loop in this second part establishes the linkage through the nodes in the list. The second sub-loop traverses the linked list using a pointer variable.

We have already loaded the program TAKF51L.AOM into the *iceMASTER* emulator (TAKF51L.AOM: Test/program A, Keil/Franklin compiler, MCS-51 chip architecture, Large memory model, Absolute Object Module format). We have also opened the Watch Window and set several Watch Expressions:



Watch	Value
X:008E uint ai	1
X:0090 uint aj	2
X:0085 uchar ary[ai]	0xFF .. 255
X:0084 [10] uchar ary	[10] uchar
X:0084 uchar ..	[0] 0xFF .. 255
X:0085 uchar ..	[1] 0xFF .. 255
X:0086 uchar ..	[2] 0xFF .. 255
X:0087 uchar ..	[3] 0xFF .. 255
X:0088 uchar ..	[4] 0xFF .. 255
X:0089 uchar ..	[5] 0xFF .. 255
X:008A uchar ..	[6] 0xFF .. 255
X:008B uchar ..	[7] 0xFF .. 255
X:008C uchar ..	[8] 0xFF .. 255
X:008D uchar ..	[9] 0xFF .. 255

The program is about to step through the array *ary* to initialize each element. There are four Watch Expressions in the Watch Window. *ai* and *aj* are simple **unsigned** integer variables at locations 0x008E and 0x0090, respectively, in External Data memory. Their current values are 1 and 2 (from source line #64). The third Watch Expression, *ary[ai]* monitors a single element in the array *ary*, but the element being monitored changes as the value of the subscript *ai* changes. The fourth Watch Expression, *ary*, watches the entire array. The "value" shown for *ary* is its data type: "[10]uchar", meaning that it is an array of 10 elements, each of which is an **unsigned char**. Following this header line for the array, each element in the array appears. The current value in each element is displayed in hexadecimal, character and unsigned decimal formats, preceded by the element's actual subscript.

Let's single-step the program by one HLL (Higher-Level Language) source line:

The screenshot shows the TI-89 Emulator interface. The assembly code window displays the following lines of assembly:

```

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help
Module:TA Source File:takf511.aom
75      for ( ai = 0; ai < ARYSIZ; ai++ ) {
76          aj = (ai * 2) + main_loop_factor;
77          ary[ai] = aj;
78      } /*for ai*/

```

The Watch window shows the following memory dump:

Address	Type	Value
X:008E	uint	ai
X:0090	uint	aj
X:0084	uchar	ary[ai]
X:0084	[10]uchar	ary
X:0084	uchar
X:0085	uchar
X:0086	uchar
X:0087	uchar
X:0088	uchar
X:0089	uchar
X:008A	uchar
X:008B	uchar
X:008C	uchar
X:008D	uchar

The value at X:0084 (ary[ai]) is highlighted in yellow, showing the value 0xFF. The value at X:008E (ai) is also highlighted in yellow, showing the value 2.

Begin emulation
 sLoad SetBkpt ViewTrc DisAssm MacExec CurMod SrcPath RawSrc SymGlob1 SymAlph
 1Help 2ResetEm3ResetTg4Go 5GoFrom 6GoUntil7StepIns8StepLin9StepOvr0StepTo

The program just executed the for loop initialization statement ($ai = 0$); and is about to execute the first line in the for loop. Notice that the value of ai has changed from 1 to 0; it is highlighted to notify you that the value changed. Also note that the address for the Watch Expression $ary[ai]$ has changed (because the subscript value ai changed); it, too, is highlighted to flag the change.

Now, let's step twice to get past the first array element assignment at line #77 at the end of the loop:

The screenshot shows the TI-89 Emulator interface. The assembly code window displays the same lines of assembly as before:

```

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help
Module:TA Source File:takf511.aom
75      for ( ai = 0; ai < ARYSIZ; ai++ ) {
76          aj = (ai * 2) + main_loop_factor;
77          ary[ai] = aj;
78      } /*for ai*/

```

The Watch window shows the following memory dump after a step operation:

Address	Type	Value
X:008E	uint	ai
X:0090	uint	aj
X:0084	uchar	ary[ai]
X:0084	[10]uchar	ary
X:0084	uchar
X:0085	uchar
X:0086	uchar
X:0087	uchar
X:0088	uchar
X:0089	uchar
X:008A	uchar
X:008B	uchar
X:008C	uchar
X:008D	uchar

The value at X:0084 (ary[ai]) is highlighted in yellow, showing the value 0x01. The value at X:008E (ai) is highlighted in yellow, showing the value 1.

Begin emulation
 sLoad SetBkpt ViewTrc DisAssm MacExec CurMod SrcPath RawSrc SymGlob1 SymAlph
 1Help 2ResetEm3ResetTg4Go 5GoFrom 6GoUntil7StepIns8StepLin9StepOvr0StepTo

After executing the assignment $ary[ai] = aj$; we see that the value stored in the first element (subscript value zero) has changed, and the corresponding change is reflected in the copy being monitored by Watch Expression $ary[ai]$.

Let's take several more trips through the loop:

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help

Module: TA

Source

File:takf511.aom

```
75     for ( ai = 0; ai < ARYSIZ; ai++ ) {
76         aj = (ai * 2) + main_loop_factor;
77         ary[ai] = aj;
78     } /*for ai*/
```

Watch

X:008E	uint	ai	4
X:0090	uint	aj	7
X:0088	uchar	ary[ai]	0xFF .. 255
X:0084	[10]uchar	ary	[10]uchar
X:0084	uchar		[0] 0x01 .. 1
X:0085	uchar		[1] 0x03 .. 3
X:0086	uchar		[2] 0x05 .. 5
X:0087	uchar		[3] 0x07 .. 7
X:0088	uchar		[4] 0xFF .. 255
X:0089	uchar		[5] 0xFF .. 255
X:008A	uchar		[6] 0xFF .. 255
X:008B	uchar		[7] 0xFF .. 255
X:008C	uchar		[8] 0xFF .. 255
X:008D	uchar		[9] 0xFF .. 255

Begin emulation

sLoad SetBkpt ViewTrc DisAssm MacExec CurMod SrcPath RawSrc SymGlob SymAlpha
1Help 2ResetEm 3ResetTg 4Go 5GoFrom 6GoUntil 7StepIns 8StepLin 9StepOvr 0StepTo

We're again at the first statement in the loop after making four complete passes through the loop. The first four elements in the array have been initialized and the subscript *ai* is set for the fifth element in the array. Now let's step to the end of the loop one more time:

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help

Module: TA

Source

File:takf511.aom

```
75     for ( ai = 0; ai < ARYSIZ; ai++ ) {
76         aj = (ai * 2) + main_loop_factor;
77         ary[ai] = aj;
78     } /*for ai*/
```

Watch

X:008E	uint	ai	4
X:0090	uint	aj	9
X:0088	uchar	ary[ai]	0x09 .. 9
X:0084	[10]uchar	ary	[10]uchar
X:0084	uchar		[0] 0x01 .. 1
X:0085	uchar		[1] 0x03 .. 3
X:0086	uchar		[2] 0x05 .. 5
X:0087	uchar		[3] 0x07 .. 7
X:0088	uchar		[4] 0x09 .. 9
X:0089	uchar		[5] 0xFF .. 255
X:008A	uchar		[6] 0xFF .. 255
X:008B	uchar		[7] 0xFF .. 255
X:008C	uchar		[8] 0xFF .. 255
X:008D	uchar		[9] 0xFF .. 255

Begin emulation

sLoad SetBkpt ViewTrc DisAssm MacExec CurMod SrcPath RawSrc SymGlob SymAlpha
1Help 2ResetEm 3ResetTg 4Go 5GoFrom 6GoUntil 7StepIns 8StepLin 9StepOvr 0StepTo

Let's move on to the second part of the program.

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help
 Module: TA Source File:takf511.aom

```

91     p_list = list_head;
92     while (p_list != NULL) {
93         l1j      = p_list->list_uival;
94         p_list = p_list->list_p_nxt;
95     } /*while p_list*/
  
```

Watch

X:00A8 ->struct List	p_list	?:FFFF{FFFFFF}
C:FFFF?struct List	*p_list	struct List
C:FFFF?uint	list_uival 65282
C:0001?->struct List	list_p_nxt C:CEFF
X:0092 [3]struct List	nodes	[3]struct List
X:0092 struct List	[0] struct List
X:0092 uint	list_uival 1
X:0094 ->struct List	list_p_nxt (null)
X:0097 struct List	[1] struct List
X:0097 uint	list_uival 5
X:0099 ->struct List	list_p_nxt X:0092
X:009C struct List	[2] struct List
X:009C uint	list_uival 9
X:009E ->struct List	list_p_nxt X:0097

Begin emulation

1Help 2ResetEm3ResetTg4Go

5GoFrom 6GoUntil7StepIns8StepLine9StepOver10StepTo

At this point, the program has executed the first sub-loop that establishes the list linkage through the structure nodes in the array *nodes* (an array of **structures**). The list head pointer, *list_head*, has been set to point at the first node in the list. That node is the last element in the array (subscript value [2], at address 0x009C in External Data memory), as the list was linked "backwards" through the array elements. The list is linked using the *list_p_nxt* field in each node.

The Watch Window contains three Watch Expressions.

The variable *p_list* is a pointer to a LIST node and it currently contains an illegal pointer value. That's fine because the program has not yet assigned a value to *p_list*. A pointer such as *p_list* occupies 3 bytes in memory. The first byte contains a selector value that indicates which memory space (e.g, Code memory, External Data memory, etc.) is being referenced by the rest of the pointer. The other two bytes in the pointer contain the offset into that memory space. In this case, the memory space selector value is invalid (0xFF), so the software displays it's code as a question mark (?) followed by a colon (:) and the offset bytes in hexadecimal. In addition, because the pointer is illegal, all three "raw" bytes in the pointer's value are displayed in hexadecimal within braces.

The second Watch Expression, **p_list*, is much like the Watch Expression *ary[ai]* used in the earlier part of this tutorial. The expression **p_list* designates "the object pointed to by the pointer *p_list*" (C indirection). In this case, the object is a **structure** of type **struct List**. That structure has two members: an unsigned integer *list_uival* and a pointer, *list_p_nxt*, to another instance of the same structure type. Note the question marks (?) between the address and data type name fields for the **structure** denoted by the Watch Expression **p_list*. The host software is flagging this object specially because it had to "follow" an invalid or NULL pointer to get to the object.

The third Watch Expression is *nodes*, which is the array of **structures** containing the nodes in the linked list.

Let's execute the highlighted line to initialize *p_list* so that it points at the first node in the chain:

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help
Module: TA Source File:takf511.aom

```

91     p_list = list_head;
92     while (p_list != NULL) {
93         l1      = p_list->list_uival;
94         p_list = p_list->list_p_nxt;
95     } /*while p_list*/

```

Watch

X:00A8 ->struct List p_list	X:009C
X:009C struct List *p_list	struct List
X:009C uint	list_uival 9
X:009E ->struct List	list_p_nxt X:0097
X:0092 [3]struct List nodes	[3]struct List
X:0092 struct List	[0] struct List
X:0092 uint	list_uival 1
X:0094 ->struct List	list_p_nxt (null)
X:0097 struct List	[1] struct List
X:0097 uint	list_uival 5
X:0099 ->struct List	list_p_nxt X:0092
X:009C struct List	[2] struct List
X:009C uint	list_uival 9
X:009E ->struct List	list_p_nxt X:0097

Begin emulation

1Help 2ResetEm3ResetTg4Go 5GoFrom 6GoUntil7StepIns8StepLine9StepOver10StepTo

Notice that *p_list* now contains a valid pointer value and that the node denoted by the expression **p_list* is the first node in the linked list. Let's step through to the end of the first pass in this loop:

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help
Module: TA Source File:takf511.aom

```

91     p_list = list_head;
92     while (p_list != NULL) {
93         l1      = p_list->list_uival;
94         p_list = p_list->list_p_nxt;
95     } /*while p_list*/

```

Watch

X:00A8 ->struct List p_list	X:0097
X:0097 struct List *p_list	struct List
X:0097 uint	list_uival 5
X:0099 ->struct List	list_p_nxt X:0092
X:0092 [3]struct List nodes	[3]struct List
X:0092 struct List	[0] struct List
X:0092 uint	list_uival 1
X:0094 ->struct List	list_p_nxt (null)
X:0097 struct List	[1] struct List
X:0097 uint	list_uival 5
X:0099 ->struct List	list_p_nxt X:0092
X:009C struct List	[2] struct List
X:009C uint	list_uival 9
X:009E ->struct List	list_p_nxt X:0097

Begin emulation

1Help 2ResetEm3ResetTg4Go 5GoFrom 6GoUntil7StepIns8StepLine9StepOver10StepTo

p_list now points at the second node in the chain and the updated Watch Expression **p_list* shows a highlighted copy of that node.

Finally, let's step until very end of the loop:

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help
 Module: TA Source File:takf511.aom

```

91     p_list = list_head;
92     while (p_list != NULL) {
93         l1 = p_list->list_uival;
94         p_list = p_list->list_p_nxt;
95     } /*while p_list*/
  
```

Watch

X:00A8 ->struct List	p_list	(null)
C:0000?struct List	*p_list	struct List
C:0000?uint	list_uival 517
C:0002?->struct List	list_p_nxt ?:FFFF{CEFFFF}
X:0092 [3]struct List	nodes	[3]struct List
X:0092 struct List	[0] struct List
X:0092 uint	list_uival 1
X:0094 ->struct List	list_p_nxt (null)
X:0097 struct List	[1] struct List
X:0097 uint	list_uival 5
X:0099 ->struct List	list_p_nxt X:0092
X:009C struct List	[2] struct List
X:009C uint	list_uival 9
X:009E ->struct List	list_p_nxt X:0097

Begin emulation 1Help 2ResetEm3ResetTg4Go 5GoFrom 6GoUntil7StepIns8StepLin9StepOut?StepTo

Here we see that *p_list* is NULL, as expected. Also note that the object denoted by the expression **p_list* is again flagged as "bad" because the *iceMASTER* host software followed a NULL pointer to get to it.

Browsing

You can "jump into" the Watch Window to browse/inspect/change the values in objects designated by the Watch Expressions. To do this, select the *Configure|Windows|Goto* command (shortcut: press the Tab key). As you scroll up or down, if the currently highlighted item is a scalar (integral or floating-point) or a pointer, you will see the *Enter:Change* prompt appear on the top-left border of the Watch Window. When you press the *Enter* key, a Change Box pops up to allow you to change the value stored in the highlighted item. A Change Box shows the address, size and data type of the object, as well as its current value. In this case, we are given the opportunity to change a member (*list_uival*) in a structure (which is itself an element in an array):

Module: TA Source File:takf511.aom

```

91     p_list = list_head;
92     while (p_list != NULL) {
93         l1 = p_list->list_uival;
94         p_list = p_list->list_p_nxt;
95     } /*while p_list*/
  
```

Enter:Change Watch

X:00A8 ->struct List	p_list	(null)
C:0000?struct List	*p_list	struct List
C:0000?uint	list_uival 517
C:0002?->struct List	list_p_nxt ?:FFFF{CEFFFF}
X:0092 [3]struct List	nodes	[3]struct List
X:0092 struct List	[0] struct List
X:0092 uint	list_uival 1
		list_p_nxt (null)
		[1] struct List
		list_uival 5
		list_p_nxt X:0092
		[2] struct List
		list_uival 9
		list_p_nxt X:0097

Change Value (Content)
 Addr: X:0092 Size: 2 bytes Type: uint
 Content
 Current: 1
 New:
 Esc: Do not change current value

Tab/Shift-Tab:Next/Prev window (Keypad):Scroll Esc:Exit
 1Help 2ResetEm3ResetTg4Go 5GoFrom 6GoUntil7StepIns8StepLin9StepOut?StepTo

If the currently highlighted item is a pointer, structure, union or array, you will see the **Ctrl-B:Browse** prompt appear on the top-right border of the Watch Window. This means that you can "browse" (inspect and/or change) the designated object. Here we'll press **Ctrl-B** to pop up a Browse Window to look at the node pointed at by the link field (*list p nxt*) in the first node in the list (that is, we'll "follow" the pointer):

Module:TA Source File:takf511.aom

```
91     p_list = list_head;
92     while (p_list != NULL) {
93         l1      = p_list->list_uival;
94         p_list = p_list->list_p_nxt;
95     } /*while p_list*/
```

Enter:Change Watch Ctrl-B:Browse

X:00A8	->struct List	p_list	(null)
C:0000		X:009E	->struct List
C:0000	Addr	Data Type	Value
C:0002			
X:0092	X:0097	struct List	struct List
X:0092	X:0097	uint	list_uival 5
X:0092	X:0099	->struct List	list_p_nxt X:0092
X:0094			
X:0097	struct List	[1] struct List
X:0097	uint	list_uival 5
X:0099	->struct List	list_p_nxt X:0092
X:009C	struct List	[2] struct List
X:009C	uint	list_uival 9
X:009E	->struct List	list_p_nxt X:0097

A Browse Window shows the entire object denoted by the pointer. A Browse Window is scrollable, moveable and resizeable. Thus, you can inspect large objects with no problem. You can even write the entire contents of a Browse Window to a file of your choice. Let's move the highlight bar to the link field (*list p_nxt*) in the **struct** we are inspecting:

Module:TA Source File:takf511.aom

```
91     p_list = list_head;
92     while (p_list != NULL) {
93         l1     = p_list->list_uival;
94         p_list = p_list->list_p_nxt;
95     } /*while p_list*/
```

Enter:Change Watch Ctrl-B:Browse

X:00A8	->struct List	p_list	(null)
C:0000	->X:009E	->struct List	t List
C:0000	Addr	Data Type	Value
C:0002	X:0092	struct List	struct List
X:0092	X:0097	uint	list_uival 5
X:0092	X:0099	->struct List	list_p_nxt X:0092
X:0094	Enter:Change	Ctrl-B:Browse	list_p_nxt (null)
X:0097	struct List	[1] struct List
X:0097	uint	list_uival 5
X:0099	->struct List	list_p_nxt X:0092
X:009C	struct List	[2] struct List
X:009C	uint	list_uival 9
X:009E	->struct List	list_p_nxt X:0097

Tab/Shift-Tab:Next/Prev window (Keypad):Scroll Esc:Exit
Help 2ResetEm3ResetTa4Go 5GoFrom 6GoUntil 7StepIns 8StepIn 9StepOut 10StepTo

Notice that the bottom border of the Browse Window shows that we can change this pointer value (**Enter:Change** on the left) or we can follow it (**Ctrl-B:Browse** on the right). Let's follow it to the next node in the list.

Module: TA Source File:takf511.aom

```

91     p_list = list_head;
92     while (p_list != NULL) {
93         l1j = p_list->list_uival;
94         p_list = p_list->list_p_nxt;
95     } /*while p_list*/

```

Enter:Change Watch Ctrl-B:Browse

X:00A8 ->struct List p_list	(null)		
C:0000	X:009E ->struct List	list	
C:0002	Addr	X:0099 ->struct List	517
C:0002	Addr	Data Type	?:FFFF{CEFFFF}
X:0092	X:0097	struct List	st
X:0092	X:0097	struct List	List
X:0092	X:0099	uint	list_uival 1
X:0094	Enter	->struct List	p_nxt (null)
X:0097	struct	list_p_nxt (null)	p_nxt (null)
X:0097	uint	Change Value (Content)	
X:0099	->struc	Addr: X:0094 Size: 3 bytes Type: ->struct List	Content
X:009C	struct	Current: (null)	
X:009C	uint	New: X:0x009C	
X:009E	->struc	Esc: Do not change current value	

Tab/Shift-Tab:Next/Prev window (Keypad):Scroll Esc:Exit
 1Help 2ResetEm3ResetTg4Go 5GoFrom 6GoUntil7StepIns8StepLin9StepOver10StepTo

Here we see that we are at the last node in the linked list (the link field `list_p_nxt` is NULL). Let's move to that field and change the link to make the list circular. We'll set the link field to point at location 0x009C in External Data memory. That is the location of the head (first) node in the chain.

Now if we continue to follow the link field (repetitively pressing **Ctrl-B** while the highlight bar is on the member `list_p_nxt`), we'll go on, and on, and on ...

Module: TA Source File:takf511.aom

```

91     p_list = list_head;
92     while (p_list != NULL) {
93         l1j = p_list->list_uival;
94         p_list = p_list->list_p_nxt;
95     } /*while p_list*/

```

Enter:Change Watch Ctrl-B:Browse

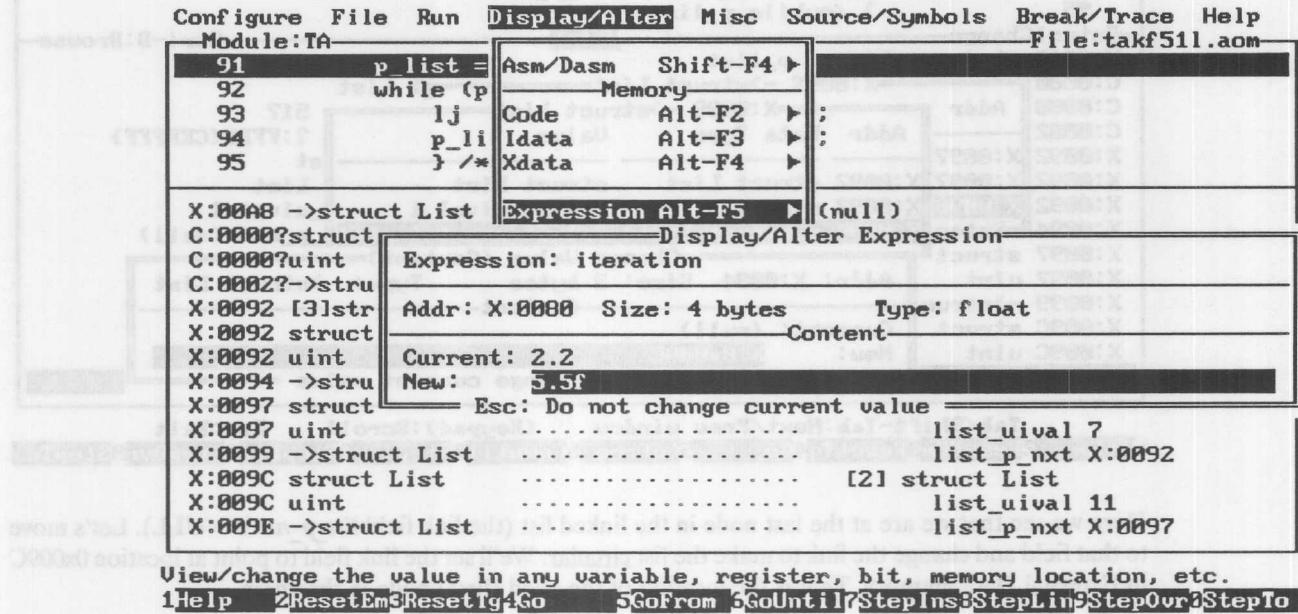
X:00A8 ->struct List p_list	(null)		
C:0000	X:009E ->struct List	list	
C:0002	Addr	X:0099 ->struct List	517
C:0002	Addr	X:0094 ->struct List	?:FFFF{CEFFFF}
X:0092	X:0097	Addr	X:009E ->struct List
X:0092	X:0097	Addr	X:0099 ->struct List
X:0094	->struct List	Data Type	Value
X:009C	Addr	X:0099 ->struct List	struct List
X:009C	X:0097	Addr	list_uival 1
X:009E	X:0097	Addr	list_p_nxt X:009C
Enter	X:0099	X:0092	Ctrl-B:Browse
Enter	X:0099	X:0092	list_uival 9
X:009	Enter	X:0094 ->struct List	list_p_nxt X:0097

Tab/Shift-Tab:Next/Prev window (Keypad):Scroll Esc:Exit
 1Help 2ResetEm3ResetTg4Go 5GoFrom 6GoUntil7StepIns8StepLin9StepOver10StepTo

Enough.

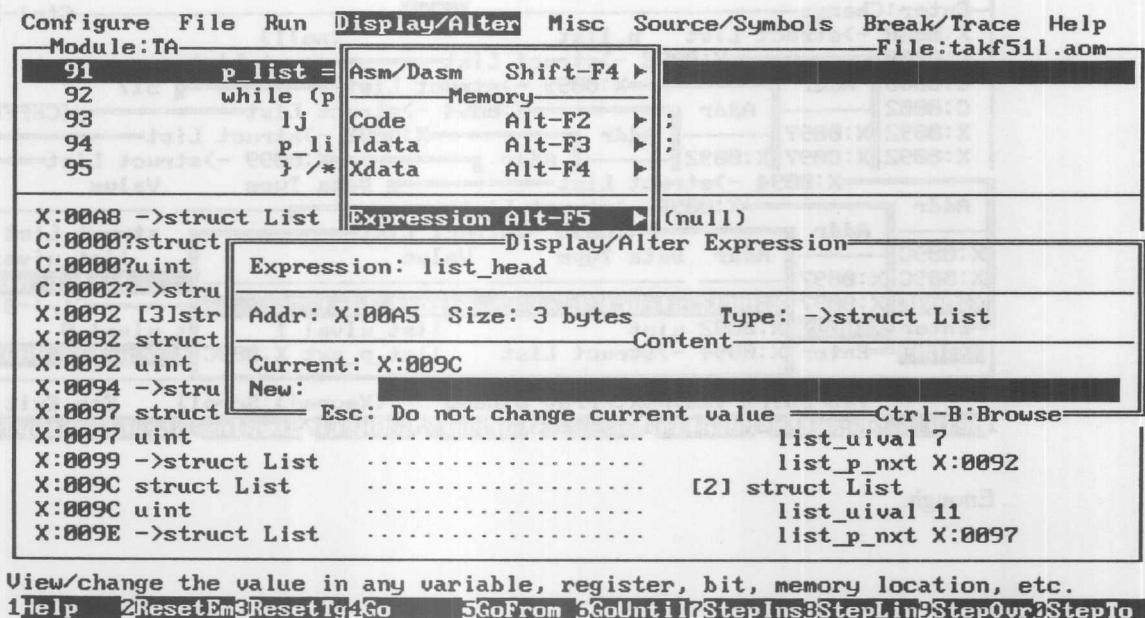
Display/Alter | Expression Command

The *Display/Alter | Expression* command pops up a dialog box which allows you to view or change the value stored in a location designated by an arbitrary expression:



The dialog box shows the address, size (in bytes or bits) and data type of the object denoted by the expression you type (in this case, the variable *iteration*). Additionally, it shows the current value of the expression (2.2 in this case), and gives you the opportunity to assign a new value to the location designated by the expression. Here we are changing the value/content of *iteration* from 2.2 to 5.5).

If the expression has type "pointer to ...", you can either change the pointer's value by entering a new value or follow the pointer (the **Ctrl-B:Browse** prompt appears on the bottom-right border of the box):



If we choose to follow it by pressing **Ctrl-B**, again a Browse Window pops up to let us do just that:

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help
Module:TA File:takf511.aom

```

91     p_list = Asm/Dasm Shift-F4 ▶
92     while (p     Memory
93         lj     Code     Alt-F2 ▶;
94         p_li  Idata    Alt-F3 ▶;
95     } /* Xdata    Alt-F4 ▶
X:00A8 ->struct List Expression Alt-F5 ▶ (null)
C:0000?struct Display/Alter Expression
C:0000?uint Expression: list_head
C:0002?->stru X:00A5 ->struct List list_head=
X:0092 [3]stru Addr Data Type Value
X:0092 struct
X:0092 uint X:009C struct List struct List
X:0094 ->stru X:009C uint list_uival 11
X:0097 struct X:009E ->struct List list_p_nxt X:0097
X:0097 uint Ctrl-R:Resize Ctrl-V:Move
X:0099 ->struct List ..... list_p_nxt X:0092
X:009C struct List ..... [2] struct List
X:009C uint ..... list_uival 11
X:009E ->struct List ..... list_p_nxt X:0097

```

View/change the value in any variable, register, bit, memory location, etc.
 1Help 2ResetEm3ResetTg4Go 5GoFrom 6GoUntil7StepIns8StepLin9StepOver10StepTo

If the expression has type "array of ...", the *Display/Alter|Expression* command shows us the size of the entire array *nodes* (15 bytes in this) case and prompts us that we can browse (inspect) the array (Ctrl-B:Browse prompt on the bottom-right border of the box):

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help
Module:TA File:takf511.aom

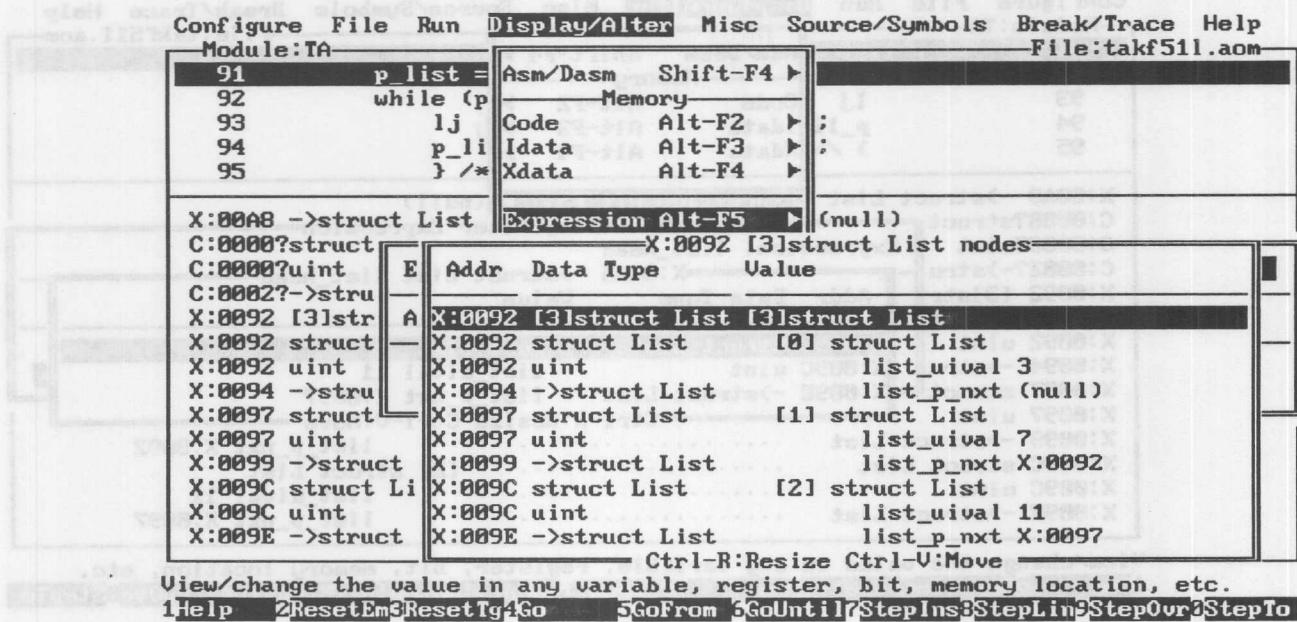
```

91     p_list = Asm/Dasm Shift-F4 ▶
92     while (p     Memory
93         lj     Code     Alt-F2 ▶;
94         p_li  Idata    Alt-F3 ▶;
95     } /* Xdata    Alt-F4 ▶
X:00A8 ->struct List Expression Alt-F5 ▶ (null)
C:0000?struct Display/Alter Expression
C:0000?uint Expression: nodes
C:0002?->stru X:0092 Size: 15 bytes Type: [3]struct List
X:0092 struct
X:0092 uint
X:0094 ->stru X:0097 struct "<expr>" or "<expr> <new value>" Ctrl-B:Browse
X:0097 uint ..... list_uival 7
X:0099 ->struct List ..... list_p_nxt X:0092
X:009C struct List ..... [2] struct List
X:009C uint ..... list_uival 11
X:009E ->struct List ..... list_p_nxt X:0097

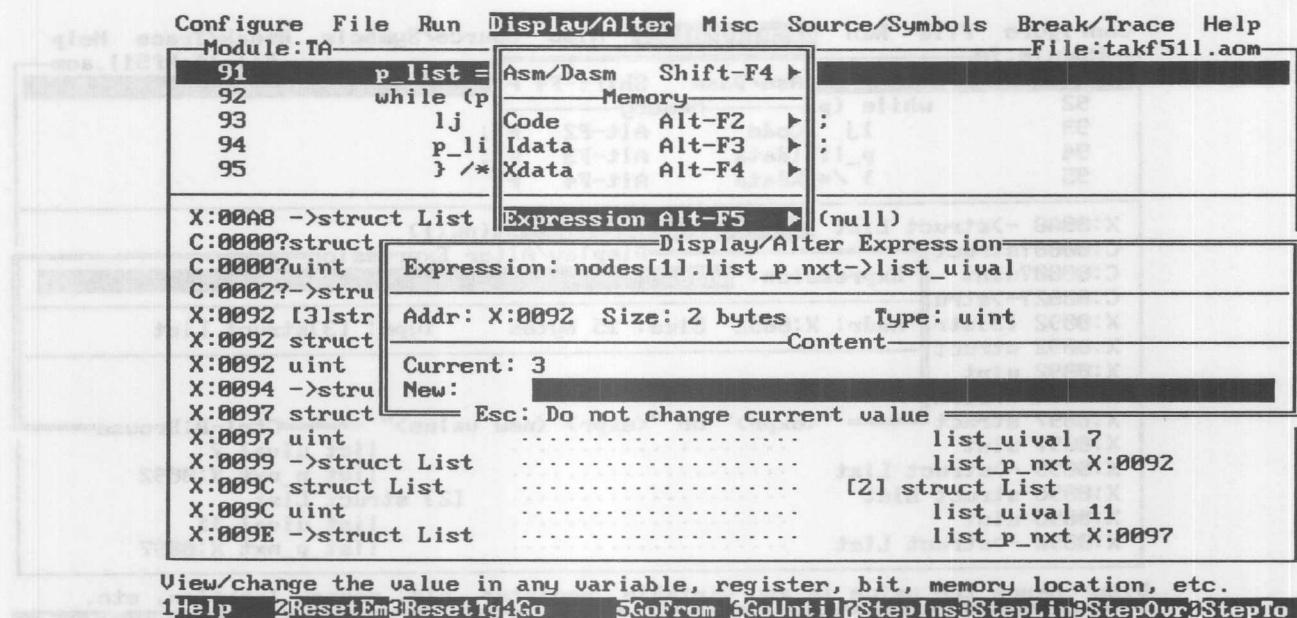
```

View/change the value in any variable, register, bit, memory location, etc.
 1Help 2ResetEm3ResetTg4Go 5GoFrom 6GoUntil7StepIns8StepLin9StepOver10StepTo

Pressing Ctrl-B pops-up a Browse Window, letting us inspect and/or change the individual elements in the array:



You can enter almost any expression in the *Display/Alter|Expression* dialog box, designating array elements and struct/union members just as you would in your C program:



Or you can follow a pointer through several levels of indirection in one step:

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help
 Module: TA File:takf511.aom

91 p_list =	Asm/Dasm Shift-F4 ►	Memory
92 while (p	Code Alt-F2 ►	:
93 l_j	Idata Alt-F3 ►	:
94 p_li	Xdata Alt-F4 ►	:
95 } /*		
X:00A8 ->struct List		
C:0000?struct		
C:0000?uint		
C:0002?->stru		
X:0092 [3]str		
X:0092 struct		
X:0092 uint		
X:0094 ->stru		
X:0097 struct		
X:0097 uint		
X:0099 ->struct List		
X:009C struct List		
X:009C uint		
X:009E ->struct List		
Expression Alt-F5 ► (null)		
Display/Alter Expression		
Expression: list_head->list_p_nxt->list_p_nxt		
Addr: X:0099 Size: 3 bytes Type: ->struct List		
Content		
Current: X:0092		
New: Esc: Do not change current value — Ctrl-B:Browse		
list_uival 7		
list_p_nxt X:0092		
[2] struct List		
list_uival 11		
list_p_nxt X:0097		

View/change the value in any variable, register, bit, memory location, etc.
 1Help 2ResetEm3ResetTg4Go 5GoFrom 6GoUntil7StepIns8StepLin9StepOvr0StepTo

And, of course, you can also monitor such complicated expressions in the Watch Window.

Finally, in the simplest case, the *Display/Alter|Expression* command even operates as a calculator:

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help
 Module: TA File:takf511.aom

91 p_list =	Asm/Dasm Shift-F4 ►	Memory
92 while (p	Code Alt-F2 ►	:
93 l_j	Idata Alt-F3 ►	:
94 p_li	Xdata Alt-F4 ►	:
95 } /*		
X:00A8 ->struct List		
C:0000?struct		
C:0000?uint		
C:0002?->stru		
X:0092 [3]str		
X:0092 struct		
X:0092 uint		
X:0094 ->stru		
X:0097 struct		
X:0097 uint		
X:0099 ->struct List		
X:009C struct List		
X:009C uint		
X:009E ->struct List		
Expression Alt-F5 ► (null)		
Display/Alter Expression		
Expression: 55u+66u		
Hex: 0x0079 Size: 2 bytes Type: uint		
Value		
Current: 121		
<expr> or <expr> <new value>		
list_uival 7		
list_p_nxt X:0092		
[2] struct List		
list_uival 11		
list_p_nxt X:0097		

View/change the value in any variable, register, bit, memory location, etc.
 1Help 2ResetEm3ResetTg4Go 5GoFrom 6GoUntil7StepIns8StepLin9StepOvr0StepTo

Source/Symbols Commands

The *Source/Symbols* commands display the symbols in your program sorted in various ways. Here we see the local symbols in TAKF51L.AOM sorted alphabetically within the module in which they were defined:

Local Symbols - By Module					
Enter Symbol					(15 Symbols)
Symbol Name	Addr	(M:Module F:Function)	Data Type	Value	Defined Within (#: Scope Level)
C:00D9	Array_Init		(label)		2 MAIN
C:027D	End_Loop		(label)		2 MAIN
C:0135	List_Init		(label)		2 MAIN
C:01E7	List_Scan		(label)		2 MAIN
X:008E	ai		uint	10	2 MAIN
X:0090	aj		uint	21	2 MAIN
X:0084	ary		[10]uchar	[10]uchar	2 MAIN
X:0080	iteration		float	5.5	2 MAIN
X:00A1	li		uint	3	2 MAIN
X:00A5	list_head		->struct List	X:009C	2 MAIN
X:00A3	lj		uint	2	2 MAIN
X:00AB	main_loop_factor		uint	3	2 MAIN
X:0092	nodes		[3]struct List	[3]struct List	2 MAIN
X:00A8	p_list		->struct List	(null)	2 MAIN

Source level debug and symbol displays

All of the *Source/Symbols* windows have the same basic format, showing the address of the symbol, the symbol's name, its data type, current value and where it was defined in the program. As you scroll through the window, the **Enter:Change** prompt appears on the bottom-left border of the window if you can change the currently highlighted item. This is the case for scalars (integral and floating-point) and pointers. Pressing **Enter** while *li* is highlighted pops up a Change Box to allow you to change its current value:

Local Symbols - By Module					
Enter Symbol					(15 Symbols)
Symbol Name	Addr	(M:Module F:Function)	Data Type	Value	Defined Within (#: Scope Level)
C:00D9	Array_Init		(label)		2 MAIN
C:027D	End_Loop		(label)		2 MAIN
C:0135	List_Init		(label)		2 MAIN
C:01E7	List_Scan		(label)		2 MAIN
X:008E	ai		uint	10	2 MAIN
X:0090	aj		uint	21	2 MAIN
X:0084	ary		[10]uchar	[10]uchar	2 MAIN
X:0080	iteration		float	5.5	2 MAIN
X:00A1	li		uint	3	2 MAIN
X:	Change Value (Content)			X:009C	2 MAIN
X:	Addr: X:00A1	Size: 2 bytes	Type: uint		2 MAIN
X:	Content				3 MAIN
X:	Current: 3				struct List 2 MAIN
X:	New:				(null) 2 MAIN
E:	Esc: Do not change current value				

Source level debug and symbol displays

If a symbol is an array, structure or union, the "value" shown in the *Source/Symbols* windows is the symbol's data type:

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help
Module:TA Source File:takf511.aom

Local Symbols - By Module

Enter Symbol				(15 Symbols)	
Symbol Name	Addr	(M:Module F:Function)	Data Type	Value	Defined Within (#: Scope Level)
C:00D9	Array_Init		(label)		2 MAIN
C:027D	End_Loop		(label)		2 MAIN
C:0135	List_Init		(label)		2 MAIN
C:01E7	List_Scan		(label)		2 MAIN
X:008E	ai		uint	10	2 MAIN
X:0090	aj		uint	21	2 MAIN
X:0084	ary		[10]uchar	[10]uchar	2 MAIN
X:0080	iteration		float	5.5	2 MAIN
X:00A1	li		uint	3	2 MAIN
X:00A5	list_head		->struct List	X:009C	2 MAIN
X:00A3	lj		uint	2	2 MAIN
X:00AB	main_loop_factor		uint	3	2 MAIN
X:0092	nodes		[3]struct List	[3]struct List	2 MAIN
X:00AB	p_list		->struct List	(null)	2 MAIN

Ctrl-R:Resize Ctrl-V:Move Ctrl-B:Browse

Source level debug and symbol displays

Notice that the **Ctrl-B:Browse** prompt appears on the bottom-right border of the window. You can browse (inspect) structures, unions, arrays and pointers in all the *Source/Symbols* windows. Pressing **Ctrl-B** here pops up a Browse Window, allowing us to inspect the individual elements in the array *ary*:

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help
Module:TA Source File:takf511.aom

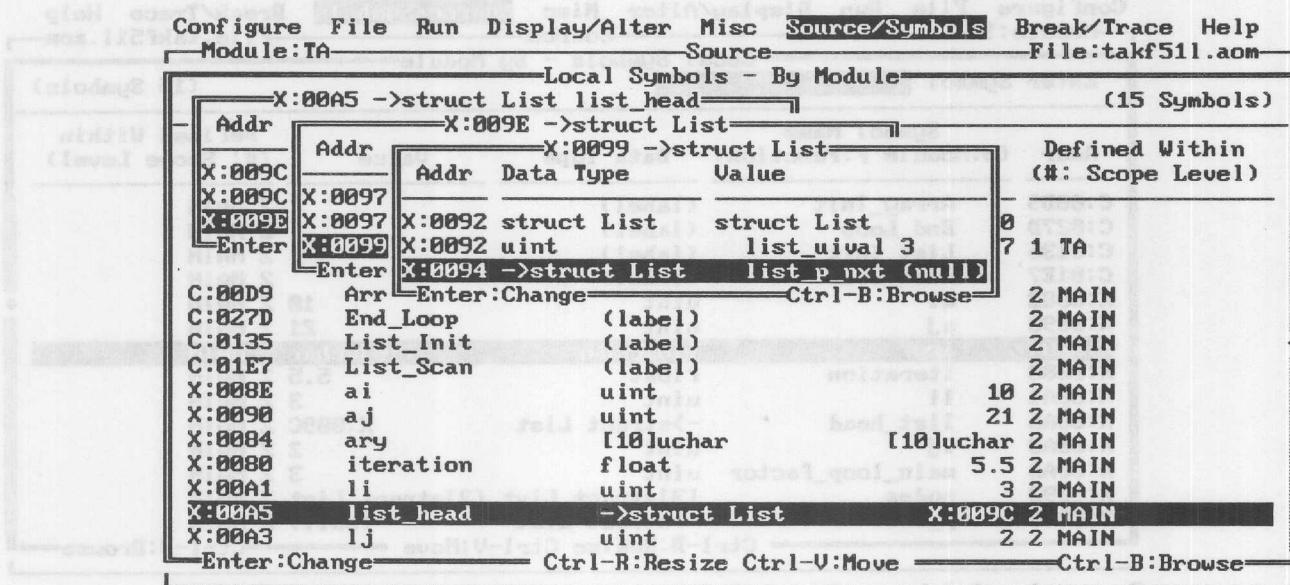
Local Symbols - By Module

X:0084 [10]uchar ary				Ctrl-W:Write	(15 Symbols)
Addr	Data Type	Value			Defined Within (#: Scope Level)
X:0084	[10]uchar	[10]uchar			
X:0084	uchar	[0] 0x03 .. 3			MAIN
X:0085	uchar	[1] 0x05 .. 5			MAIN
X:0086	uchar	[2] 0x07 .. 7			MAIN
X:0087	uchar	[3] 0x09 .. 9			MAIN
X:0088	uchar	[4] 0x0B .. 11			MAIN
X:0089	uchar	[5] 0x0D .. 13			MAIN
X:008A	uchar	[6] 0x0F .. 15			MAIN
X:008B	uchar	[7] 0x11 .. 17			MAIN
X:008C	uchar	[8] 0x13 .. 19			MAIN
X:008D	uchar	[9] 0x15 .. 21			MAIN
			Enter:Change	Ctrl-R:Resize Ctrl-V:Move	
X:00A3	lj	uint			2 2 MAIN
X:00AB	main_loop_factor	uint			3 2 MAIN
X:0092	nodes	[3]struct List	[3]struct List		2 MAIN
X:00AB	p_list	->struct List	(null)		2 MAIN

Ctrl-R:Resize Ctrl-V:Move Ctrl-B:Browse

Source level debug and symbol displays

Similarly, if the currently-highlighted symbol is a pointer to something, we can press the **Ctrl-B** key to browse that "something". And, of course, if that "something" contains elements that are browseable, we can inspect or change them, too:



Source level debug and symbol displays

Improved Support for "Unsupported" Programs

Even if your program contains limited data type information, or no data type information at all, you can still benefit from some of the new features in the *iceMASTER* software.

The demo program **DEMO.DBG** is an assembly language program assembled using MetaLink's 8051 cross-assembler. The only debug information output by this assembler (and many other assemblers, too) is symbol name, address and memory space (e.g., Code memory, External Data memory, etc.). There is no data type information.

The screenshot shows the iceMASTER interface. The assembly code window displays the following code:

```
0030 900000 START: MOU DPTR,#0000H
0033 C290 OUTERLOOP: CLR T2
0035 75640A MOU TEMPCount,#0Ah
0038 120050 INNERLOOP: LCALL WASTETIME
003B B290 CPL T2
003D E4 CLR A
003E 309001 JNB T2,SKIPOVER
0041 F4 CPL A
0042 D564F3 SKIPOVER: DJNZ TEMPCount, INNERLOOP
0045 A3 INC DPTR
0046 120050 LCALL WASTETIME
0049 120050 LCALL WASTETIME
004C 80E5 SJMP OUTERLOOP
004E 80E0 ENDOFPROMGRAM: SJMP START
0050 78FF WASTETIME: MOV R0,#0FFh
```

The Watch window shows a single entry:

D:0064 (unknown)	TEMPCount	0x0A
------------------	-----------	------

Begin emulation
sLoad SetBkpt ViewTrc DisAssm MacExec CurMod SrcPath RawSrc SymGlob1SymAlph
1Help 2ResetEm3ResetTg4Go 5GoFrom 6GoUntil7StepIns3StepLin9StepOvr0StepTo

In the picture above, we have loaded **DEMO.DBG** into the emulator and added the Watch Expression **TEMPCOUNT** to the Watch Window. Notice that the data type is "(unknown)". In such cases, the host software defaults to displaying the value in hexadecimal only, and the software must be "told" how many bytes to read (via the *Configure | Options | Expressions | Unknown data type size* command). Below, we are about to add a Watch Expression that will cause the host software to treat **TEMPCOUNT** as "true" one-byte unsigned character value.

The screenshot shows the iceMASTER interface. The configuration menu (Configure) is open, showing the following options:

- Emulator
 - Mapping
 - Code Memory
 - Xdata Memory
 - Mode #1
 - Display
 - Attributes
 - Windows
- Display
- Attributes
- Windows

The Watch window shows a single entry:

D:0064	Add Ctrl-F9	0A
--------	-------------	----

The status bar at the bottom says:

Enter any expression denoting an address.
sLoad SetBkpt ViewTrc DisAssm MacExec CurMod SrcPath RawSrc SymGlob1SymAlph
1Help 2ResetEm3ResetTg4Go 5GoFrom 6GoUntil7StepIns3StepLin9StepOvr0StepTo

As you can see below, there isn't much difference in the display in this case:

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help
 File:demo.dbg
 Source

0030 900000 START:	MOV D PTR, #0000H	D PTR=0000h
0033 C290 OUTERLOOP:	CLR T2	bit[90h]=1
0035 75640A	MOV TEMP COUNT, #0Ah	direct[64h]=00h
0038 120050 INNERLOOP:	L CALL W ASTETIME	10050
003B B290	CPL T2	
003D E4	CLR A	
003E 309001	JNB T2, SKIPOVER	
0041 F4	CPL A	
0042 D564F3 SKIPOVER:	D JNZ TEMP COUNT, INNERLOOP	10038 direct[64h]=0Ah
0045 A3	INC D PTR	
0046 120050	L CALL W ASTETIME	
0049 120050	L CALL W ASTETIME	
004C 80E5	S JMP OUTERLOOP	
004E 80E0 ENDOFPROMGRAM:	S JMP START	
0050 78FF W ASTETIME:	MOV R0, #0FFh	R0=64h

Watch

D:0064 (unknown)	TEMP COUNT	0x0A
D:0064 uchar	(uchar)TEMP COUNT	0x0A .. 10

Set up the system configuration

Load SetBkpt ViewTrc DisAssm MacExec CurMod SrcPath RawSrc SymGlob SymAlpha
 Help 2ResetEm 3ResetTg Go 5GoFrom 5GoUntil 7StepIns 8StepLin 9StepOvr 0StepTo

but the Watch Expression *(uchar)TEMP COUNT* will not be effected by changing the value for the *Configure|Options|Expressions|Unknown data type size* command. The host software will always read just one byte and treat that byte as an **unsigned** character value.

What if the program is manipulating a 16-bit value at address 0x0050 in External Data memory (high byte at 0x0050, low byte at 0x0051) and there is no symbol associated with that location? Here we are about to add a Watch Expression that will let us monitor that value:

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help
 File:demo.dbg
 Source

Emulator	►	MOU D PTR, #0000H	D PTR=0000h
Mapping		CLR T2	bit[90h]=1
Code Memory	►	MOV TEMP COUNT, #0Ah	direct[64h]=00h
Xdata Memory	►	L CALL W ASTETIME	10050
Mode	#1 ►	CPL T2	
Display		CLR A	
Attributes	►	JNB T2, SKIPOVER	
Windows	►	CPL A	
Modify	Ctrl-F6 ►	D JNZ TEMP COUNT, INNERLOOP	10038 direct[64h]=0Ah
Size	Ctrl-F7	INC D PTR	
Goto	Ctrl-F8	L CALL W ASTETIME	
Repaint	Alt-F1	L CALL W ASTETIME	
Watch		S JMP OUTERLOOP	
Add	Ctrl-F9 ►	S JMP START	
Add		MOV R0, #0FFh	R0=64h

Add Watch Expression

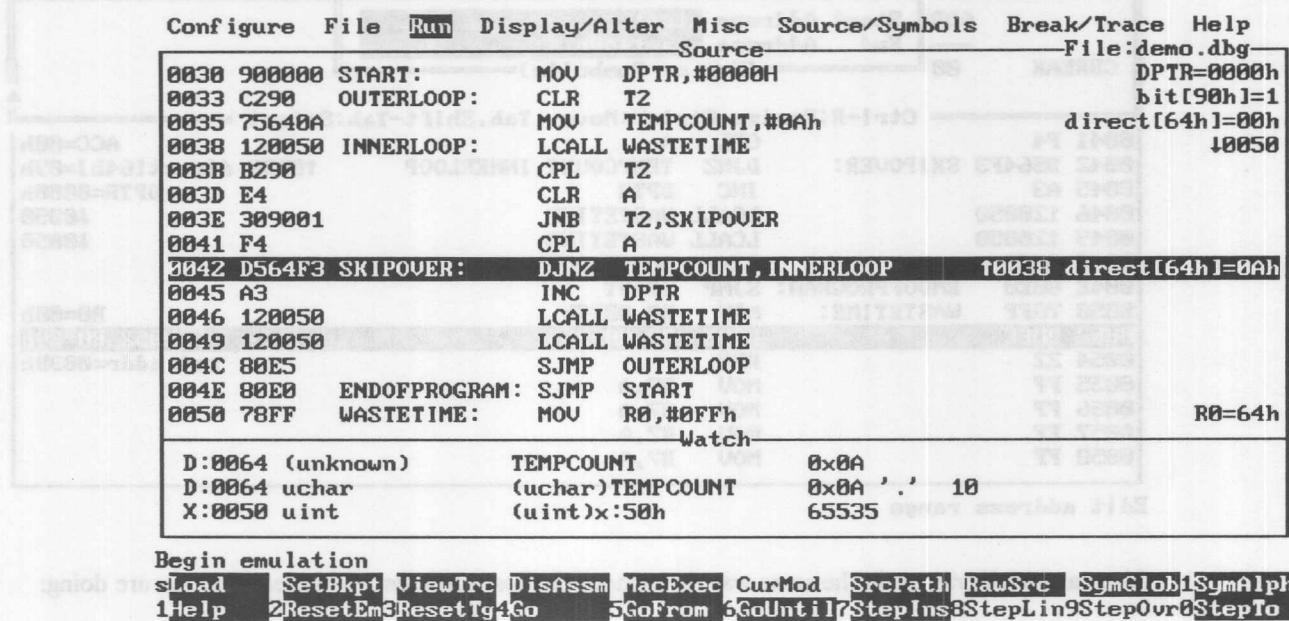
D:0064	(uint)x:50h	0A
D:0064		0A .. 10

Load ►

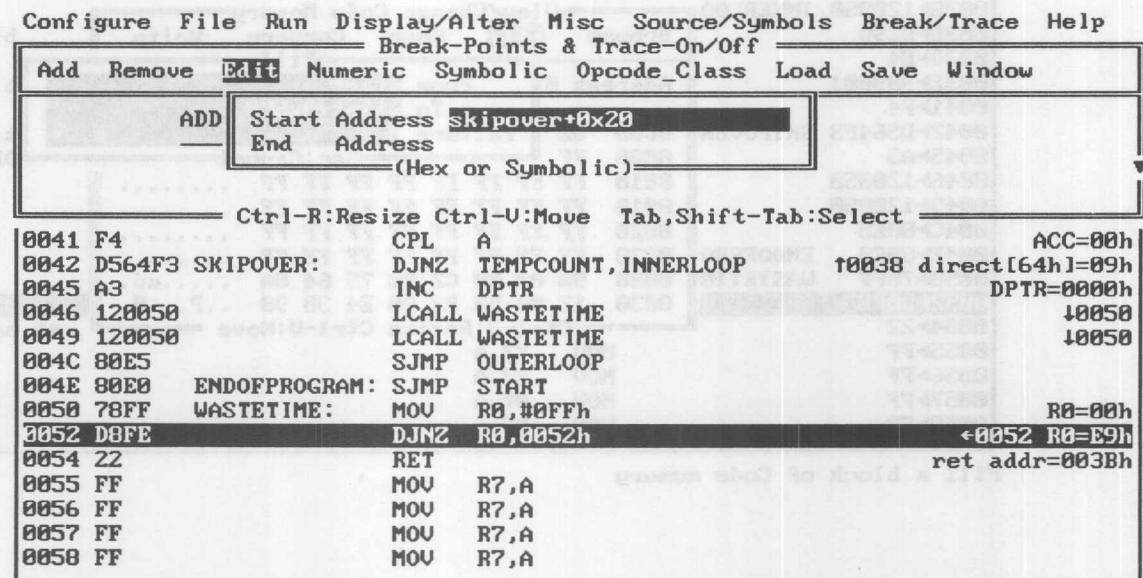
Enter any expression denoting an address.

Load SetBkpt ViewTrc DisAssm MacExec CurMod SrcPath RawSrc SymGlob SymAlpha
 Help 2ResetEm 3ResetTg Go 5GoFrom 5GoUntil 7StepIns 8StepLin 9StepOvr 0StepTo

After the Watch Expression (*uint*)*x*:50h is added, the host software will monitor the two bytes starting at location 0x0050 in External Data memory and treat those two bytes combined as a 16-bit **unsigned** integer value. The default display format for such values is (unsigned) decimal notation, although you can change that to your choice of character, decimal or hexadecimal notations using the *Configure | Options | Expressions | int/uint* command:



With the ability throughout the host software to specify addresses using expressions rather just simple labels or absolute addresses, you can, for example, set break-points relative to a label:



Edit address range

This is handy if you have a multi-module assembly language program and are using the assembler-generated listing files to set break-points. The listings contain addresses relative to the beginning of the module, not the final, absolute addresses in the bound (linked) program.

You can also set a break-point range based only on a "starting point" and a "distance", without having to compute the start and end points manually:

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help

Break-Points & Trace-On/Off

Add	Remove	Edit	Numeric	Symbolic	Opcode_Class	Load	Save	Window
CBREAK	00	ADD	Start Address	start				
			End Address	start+40h				
			(Hex or Symbolic)					

Ctrl-R:Resize Ctrl-U:Move Tab,Shift-Tab:Select

0041 F4	CPL A	ACC=00h
0042 D564F3	SKIPOVER: DJNZ TEMPCount, INNERLOOP	10038 direct[64h]=09h
0045 A3	INC D PTR	D PTR=0000h
0046 120050	LCALL WASTETIME	10050
0049 120050	LCALL WASTETIME	10050
004C 80E5	SJMP OUTERLOOP	
004E 80E0	ENDOFPROGRAM: SJMP START	
0050 78FF	WASTETIME: MOU R0, #0FFh	R0=00h
0052 D8FE	DJNZ R0, 0052h	←0052 R0=E9h
0054 22	RET	ret addr=003Bh
0055 FF	MOU R7, A	
0056 FF	MOU R7, A	
0057 FF	MOU R7, A	
0058 FF	MOU R7, A	

Edit address range

You can specify ranges in the same way throughout the host software, no matter what you are doing:

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help

Source File:demo.dbg

0030>900000	START: MOV DPTR, #0000H	D PTR=0000h
0033>C290	OUTERLOOP: CLR T2	bit[90h]=1
0035>75640A	MOV TEMPCount, #0Ah	direct[64h]=05h
0038>120050	INNERLOOP: MOU R7, A	10050
003B>B290		
003D>E4		
003E>309001		
0041>F4		
0042>D564F3	SKIPOVER: MOU R7, A	bit[90h]=0
0045>A3		ACC=00h
0046>120050		t[64h]=09h
0049>120050		D PTR=0000h
004C>80E5		
004E>80E0	ENDOFPRO MOU R7, A	
0050>78FF	WASTETIME: MOU R7, A	R0=00h
0052>D8FE		←0052 R0=E9h
0054>22		ret addr=003Bh
0055>FF		
0056>FF		
0057>FF		
0058>FF		

View/Change Code Memory

Fill a block of Code memory

From start

To start+0x500

Pattern 0

Esc:Cancel

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x500

Pattern 0

Esc:Cancel

Fill

Address 0

From start

To start+0x

Chapter 3: Reference

WordNet has entries for over 100,000 words and phrases in English. You can search for the best definition (or one of many) by clicking on the following buttons:

Search

The first option is to use the search box located at the top of the page.

Find synonyms: [S-A-X-Y-Z](#)

Find antonyms: [S-X-A-Z-B-Y](#)

Conjugate

Find conjugates

WordNet supports C-ubyx, which contains as many as a million of other pointers to other pointers. To follow a pointer, click on the word or phrase that you want to follow. A link to the original page is also provided.

verb-ger	verb-ger

The following entries contain an auxiliary verb (verb, adjective, etc.) in a box to its left. Some entries will have an auxiliary verb (verb, adjective, etc.) in the definition part, indicating that the verb is auxiliary. Following each definition are the same auxiliary verbs in boxes to its left.

aux-verb

auxiliary verb (verb, adjective, etc.)

An auxiliary verb is a verb that is used to help another verb. It is usually unnecessary to use both auxiliary verbs in a sentence.

An auxiliary verb is a verb that is used to help another verb. It is usually unnecessary to use both auxiliary verbs in a sentence.

This auxiliary verb is a verb that is used to help another verb. It is usually unnecessary to use both auxiliary verbs in a sentence.

aux

aux

auxiliary

Expression Input

Whenever and wherever the *iceMASTER* host software requests an address, count or length, you can enter an arbitrary expression (suitable for the particular context, of course). Expressions are composed of the following elements.

Identifiers

The first character in an identifier name cannot be a digit or the '\$' character:

First character: _ ? @ a-z A-Z

Subsequent characters: _ \$ a-z A-Z 0-9

Constants

Integer Constants

iceMASTER supports C-style constants as well as a variety of other popular formats. The following notations are used in subsequent definitions. A plural suffix (s) means "one or more of" the specified characters or digits. A suffix of *-opt* means that the item is optional (i.e., "zero or more of").

hex-digit: 0-9 a-f A-F

dec-digit: 0-9

non-zero-dec-digit: 1-9

oct-digit: 0-7

bin-digit: 0-1

Explicit Radix The following constants contain an explicit radix (base) designator, either as a prefix or a suffix. Such constants will always be interpreted in the designated base, regardless of the current setting of the *Configure | Options | Expressions | Radix* command (see page 3-15). Following each definition are some examples of that particular style of constant.

0x *hex-digits*

0X *hex-digits*

Hexadecimal constant with explicit prefix (C style).

An optional suffix of 'u' or 'U' specifies an unsigned integer. This is generally unnecessary, as the host software treats all constants as unsigned.

An optional suffix of 'l' or 'L' specifies a long (32-bit) integer rather than a 16-bit integer.

The suffixes 'u'/'U' and 'l'/'L' can be used at the same time, in either order.

0x12

0xa

0x12751

X' hex-digits

Hexadecimal constant with explicit prefix (NSC style).

x'ab
x'45
x'123456

\$ hex-digits

Hexadecimal constant with explicit prefix (Motorola style).

\$ab
\$0045
\$123456

dec-digit hex-digits-opt h
dec-digit hex-digits-opt H

Hexadecimal constant with explicit suffix (Intel, et. al., style).

1ah
0054h
12345678h

dec-digits t
dec-digits T
dec-digits

Decimal constant with explicit suffix.

12t
045.

non-zero-dec-digit dec-digits-opt u
non-zero-dec-digit dec-digits-opt U
non-zero-dec-digit dec-digits-opt l
non-zero-dec-digit dec-digits-opt L

Decimal constant with explicit "suffix" (C style).

A suffix of 'u' or 'U' specifies an unsigned integer. This is generally unnecessary, as the host software treats all constants as unsigned.

A suffix of 'l' or 'L' specifies a long (32-bit) integer rather than a 16-bit integer.

the suff

231

47ul

oct-digits o
oct-digits O
oct-digits q
oct-digits Q

Octal constant with explicit suffix.

12o
045o

0 oct-digits

Octal constant with explicit prefix (leading zero, C style).

An optional suffix of 'u' or 'U' specifies an unsigned integer. This is generally unnecessary, as the host software treats all constants as unsigned.

An optional suffix of 'l' or 'L' specifies a long (32-bit) integer rather than a 16-bit integer.

The suffixes 'u'/'U' and 'l'/'L' can be used at the same time, in either order.

012
055u

bin-digits y
bin-digits Y

Binary constant with explicit suffix.

1011y
011101Y

In all cases, if you do not supply an 'l' or 'L' suffix (or if such a suffix is not allowed) and the constant cannot fit in 16 bits, it will automatically be treated as a long value (32 bits).

If you are unsure of how a particular constant is being interpreted by the host software, just use the *Display/Alter|Expression* command (page 3-25) to see how the constant or constant expression is being treated.

No Explicit Radix

If you enter a number such as 10 or 101, it will be interpreted in the current default radix (base) as specified by the *Configure|Options|Expressions|Radix* command (see page 3-15). However, in certain contexts where it makes sense, the host software will temporarily override the current default. For example, when you enter a source line number as

#11

the software temporarily changes the default radix to 10 (decimal) before translating the '11'. In other cases, when the software displays a Dialog Box to obtain input, a prompt somewhere in the box (usually on the bottom border) indicates if the default radix has been changed temporarily.

In any case, you can always provide an explicit radix designator in the constant to override any default interpretation. For example, '#0x23' designates source line number 35 decimal, as does '#35t'.

In some cases, the software can automatically determine the appropriate radix based only on the characters and digits present in the constant. For example, 1A can only be the hexadecimal number 0x1A, even though no explicit prefix or suffix is present.

Floating-Point Constants

The software supports both single-precision and double-precision floating-point constants.

dec-digits . dec-digits exponent-opt
dec-digits exponent

The *exponent* has the following form:

exponent: *exp-char sign-opt dec-digits*
exp-char: **e** or **E**
sign: **+** or **-**

If there is no suffix, the constant is **double-precision floating-point (double)**. An optional suffix of '**f**' or '**F**' specifies that the constant is **single-precision floating-point (float)**. An optional suffix of '**l**' or '**L**' explicitly specifies explicitly that the constant is **double-precision floating-point (double)**.

Unlike C, "1." is not a valid floating-point constant. You must always supply at least one digit after the decimal point (e.g., "1.0").

1.2
1.2f
5.6e5

Character Constants

The value of a character constant is the numerical value of the character interpreted as an integer. Internally, character constants are treated just as if you had entered an integer constant.

'\ 1-to-3-oct-digits',
'\x 1-to-3-hex-digits',
'\ escaped-char',
'char'

The first form uses an escaped sequence of one to three octal digits.

The second form uses an escaped sequence of one to three hexadecimal digits.

The third form specifies one of the following specially recognized escape characters:

Source Form	Internal Value	Meaning
\a	0x07	ANSI alert (bell)
\b	0x08	ANSI backspace
\e	0x1B	ANSI ESC
\f	0x0C	ANSI Formfeed
\n	0x0A	ANSI New line
\r	0x0D	ANSI Carriage return
\t	0x09	ANSI Horizontal tab
\v	0x0B	ANSI Vertical tab

If the backslash precedes a character that does not appear in the table above, the value used is the value of that character itself. That is, '\w' is the same as 'w'.

The fourth form represents the standard, "simple" case.

```
'\077'  
'\xAB'  
'\n'  
'H'
```

String Constants

"*chars-opt*"

The form of each *char* in *chars-opt* is the same as that allowed for character constants. That is, all forms of escape sequences are allowed internally.

Currently, string constants can only be used as fill values in the *Display/Alter|(memory)|Fill* commands.

Expressions

The operators and operands which are allowed in expressions are grouped below in decreasing precedence (priority) order. The highest precedence operators and expression elements are listed first. To simplify the description, the term *expr* means any expression at that precedence level or higher, and the term *expression* means any expression at all.

Primary

identifier

Denotes a variable, label, module, function or SFR (Special Function Register) name. In the absence of any qualification on *identifier*, the software searches for the definition in the following order:

- local symbols in the current module
- global symbols

In the following descriptions, we use more meaningful names for identifiers to denote what kind of identifier is required in a particular context:

- *module-name*
- *function-name*
- *SFR-name*

constant

An integer constant, floating-point constant or character constant.

string

A character string constant (literal).

(*expression*)

A parenthesized expression. Parentheses can be used to change or clarify the order of operator precedence.

integer-constant

Source line number '*integer-constant*' in the current module.

module-name : # *integer-constant*

Source line number '*integer-constant*' in the module *module-name*.

module-name : *identifier*

Local variable or function *identifier* in module *module-name*.

module-name : *function-name* : *variable-name*

The local variable *variable-name* in the function *function-name* in module *module-name*.

C : *integer-constant*

X : *integer-constant*

D : *integer-constant*

I : *integer-constant*

B : *integer-constant*

Denotes a memory-space qualified constant. In most cases where the host software requests an address, you do not need to specify explicitly what memory space to use when you enter a constant or constant expression. For example, when setting a Code Memory break-point, if you enter 0x23, the host software assumes that you mean location 0x0023 in Code Memory. Similarly, if you use labels or variable names, the symbol/debug information supplied in the program load module file normally tells in which memory space that symbol belongs.

However, if you want to change the value in a generic pointer (see page 3-13), you must tell the host software the memory space in which the address exists (which memory space the address is relative to).

Designator	Memory Space	Applicability			
		MCS-51	COP8	68HC11	68HC05
C	Code	x	x	x	x
D	Directly-addressable Internal Data	x	x		
I	Indirectly-addressable Internal Data	x			
X	External Data	x			
B	Bit-addressable Internal Data	x			

You can use a memory-space qualified constant to designate an address in other contexts, too. For example, in a program with no symbolic debug information, you can add the Watch Expression X:0x23 to the Watch Window. This will allow you to monitor location 0x0023 in External Data memory.

Examples:

X:0x23 designates location 0x0023 in External Data memory

D:45h designates location 0x45 in Directly-addressable Internal Data/RAM memory

Postfix

expr . member-name

Structure or union member *member-name* in the **struct/union** expression *expr*. The *expr* must have a **struct** or **union** type.

expr . integer-constant

(MCS-51 only) Designates a particular bit in a bit-addressable SFR. *integer-constant* must be between 0 and 7, inclusive.

expr -> member-name

Structure or union member *member-name* in the **struct/union** pointed at by pointer expression *expr*. The *expr* must have the type "pointer to **struct**" or "pointer to **union**".

expr [expression]

Subscript expression. The entity being subscripted, *expr*, must have type "array of ..." or "pointer to ...". The subscript expression, *expression*, must be of an integral type.

Unary

** expr*

Indirection through pointer expression *expr*, which must be of a "pointer to ..." type.

- expr

Unary minus. The *expr* can be either integral or a floating-point constant.

+ expr

Unary plus. The *expr* can be either integral or a floating-point constant.

Cast

(type-name) expr

Casting is intended for use in conjunction with a memory-space qualified constant (an address). The meaning of such an expression is "treat the location *expr* as having data type *type-name*".

The available *type-names*, which are reserved words, are:

<i>type-name</i>	C Data Type	Size	Memory Layout
bit	bit	1 bit	
char	char	1 byte	
uchar	unsigned char	1 byte	
short	short	2 bytes	MSB, LSB
ushort	unsigned short	2 bytes	MSB, LSB
int	int	2 bytes	MSB, LSB
uint	unsigned int	2 bytes	MSB, LSB
long	long	4 bytes	MSW, LSW (each MSB, LSB)
ulong	unsigned long	4 bytes	MSW, LSW (each MSB, LSB)
float	float	4 bytes	IEEE single-precision floating-point
double	double	8 bytes	IEEE double-precision floating-point

MSB: Most Significant Byte (8 bits)

LSB: Least Significant Byte (8 bits)

MSW: Most Significant Word (16 bits, 2 bytes)

LSW: Least Significant Word (16 bits, 2 bytes)

Note that no conversions are performed on the value itself. The cast merely changes the data type that the host software associates with the designated address expression.

Multiplicative *expr* * *expr*

Multiplication.

expr / *expr*

Division.

expr % *expr*

Modulo division.

Additive *expr* + *expr*

Addition.

expr - *expr*

Subtraction.

Display Formats

This section describes the display formatting conventions used by the host software.

Addresses

Addresses displayed in the following contexts

- Watch Window
- *Display/Alter|Expression* command
- *Source/Symbols|Address* command
- *Source/Symbols|Alpha* command
- *Source/Symbols|Global* command
- *Source/Symbols|Local* command
- Pop-up Change Box
- Pop-up Browse Window

are of the form:

m:hhhh

where *m* is a character denoting the memory space and *hhhh* is the hexadecimal offset into that memory space. The possible values of *m* are:

Designator <i>m</i>	Memory Space	Applicability			
		MCS-51	COP8	68HC11	68HC05
C	Code	x	x	x	x
D	Directly-addressable Internal Data	x	x		
I	Indirectly-addressable Internal Data	x			
X	External Data	x			
B	Bit-addressable Internal Data	x			
N	(none / any / "number")	x	x	x	x

Type Names

Fundamental Types

Displayed Type Name	C Data Type	Size (Memory Layout)
(unknown)	-	<i>Configure Options Expressions Unknown data type size (see page 3-16)</i>
(untyped)	-	
bit	bit	1 bit
char	signed char	1 byte
uchar	unsigned char	1 byte
short	signed short	2 bytes (MSB, LSB)
ushort	unsigned short	2 bytes (MSB, LSB)
int	signed int	2 bytes (MSB, LSB)
uint	unsigned int	2 bytes (MSB, LSB)
long	signed long	4 bytes (MSW (MSB, LSB), LSW (MSB, LSB))
ulong	unsigned long	4 bytes (MSW (MSB, LSB), LSW (MSB, LSB))
float	float	4 bytes (IEEE format)
double	double	8 bytes (IEEE format)
void	void	-
(label)	(statement label)	-

Bit Field Types

The type name displayed for structure members which are C bit fields shows the underlying integral type preceded by the bit offset into an object of that type and the number of bits in the field. The general format is:

`offset:length_type-name`

where 'offset' is the offset to the least significant bit in the field, 'length' is the field length (in bits) and *type-name* is the name of the underlying type. The offset 'offset' is zero-relative, starting at the least significant bit. For example, the bits in a **char** are numbered 7,...,0, with 7 being the most significant bit and 0 being the least significant bit.

See page 3-18 for an example of the display of a structure containing bit field members.

Derived/Aggregate Types

The general display formats are as follows, where "... represents any data type name:

Display Format	Meaning
-> ...	"pointer to ..."
[n]...	"array of n ..."
()...	"function returning ..."
struct <i>tag-name</i>	(structure with tag name <i>tag-name</i>)
struct	(structure with no tag name)
union <i>tag-name</i>	(union with tag name <i>tag-name</i>)
union	(union with no tag name)

Unlike C, where data types read "from the inside out, obeying the parentheses", the data type names displayed by the host software read strictly from left-to-right:

Displayed Type Name	Example C Declaration	Data Type
-> int	int* pi;	"pointer to int"
[5]int	int x[5]	"array of 5 ints"
()int	int f();	"function returning int"
[15]-> int	int* y[15];	"array of 15 pointers to int"
-> ()->[10]->char	char* (*(*abc)())[10];	"pointer to a function returning a pointer to an array of 10 pointers to char"

Values

The display format of a value depends on that value's data type.

Fundamental Types

You can display values of the fundamental types in one of three formats:

- Character (enclosed in apostrophes ('))
- Decimal (signed or unsigned)
- Hexadecimal

You control the display format for a particular data type with commands in the *Configure | Options | Expressions* pull-down menu (see page 3-20).

Pointers

The display format of a pointer value depends on the "nature" of the pointer.

Generic Pointers

Generic pointers are needed due to the Harvard architecture of the MCS-51. A generic pointer occupies three bytes in memory. The first byte contains a selector value/code denoting which memory space is being referenced (e.g., Code memory, External Data memory, etc.). The next two bytes contain the offset (0x0000-0xFFFF) into that memory space.

A generic pointer is used when the compiler does not know (at compile-time) what memory space a pointer variable may reference.

The general display format for a generic pointer is:

m:hhhh

where *m* is a character denoting the memory space and *hhhh* is the hexadecimal offset into that memory space. The possible memory space characters (*m*) are a subset of those listed in the table on page 3-10.

The display format for invalid (illegal, erroneous) generic pointers is:

?:*hhhh {sshhhh}*

An invalid generic pointer value is one with an illegal memory space selector byte value. This can occur for several reasons:

- 1) The application program has not yet initialized (assigned to) the pointer.
- 2) The application program has erroneously "clobbered" (written to) the memory space selector byte value in the pointer.
- 3) The linker has assigned (static) variables to memory locations so that local variables in functions which cannot be active at the same time share the same memory for their local variables. For example, if functions 'a' and 'b' can never be active at the same time, the linker may assign the local variables in function 'a' such that they use the same memory locations as the local variables in function 'b'. If function 'b' is currently active and you display a local pointer variable defined in function 'a', the value in that pointer variable may appear to be "bad".

Memory-Specific Pointers

Memory-specific pointers are used when the compiler "knows" at compile-time what memory space is being referenced by a particular pointer. Memory-specific pointers occupy only the number of bytes required to hold an address spanning a particular memory space. Memory-specific pointers do not contain a memory space selector byte.

The general display formats for memory-specific pointers are:

(*m*):*hh*

(*m*):*hhhh*

where *m*, *hh* and *hhhh* are the same as for the generic pointers. The memory space designator *m* is included to let you know which memory space is being referenced by the offset value *hh* or *hhhh*. It is enclosed in parentheses to let you know that the memory space is implicit (i.e., there is no memory space selector byte stored in memory).

NULL Pointers

The display format for a NULL pointer value is "(null)". The rules for detecting a NULL pointer value depend on the compiler you are using. See page 3-29 for specific details about NULL pointer values.

The single-line "value" displayed for a structure, union or array object is the data type name itself ("struct...", "union..." or "[n]...").

In the main-screen Watch Window and the pop-up Browse Windows, structures, unions and arrays are fully-expanded to show the individual members or elements. In these contexts, the individual members in a **struct/union** and the individual elements in an array immediately follow the single-line header for the structure, union or array. Each of these lines show the address, data type and value of the **struct/union** member or array element.

Functions

The value displayed for an entity of type "function returning..." is the name of the function itself. This may seem redundant and silly in the simple case, but if you are following (indirecting through) a pointer to a function, the software displays the name of the function pointed to by the pointer.

... a function pointer to the function "f" is displayed as "f".

(f)(int)

... a function pointer to the function "f" is displayed as "f".

... a function pointer to the function "f" is displayed as "f".

... a function pointer to the function "f" is displayed as "f".

... a function pointer to the function "f" is displayed as "f".

f()

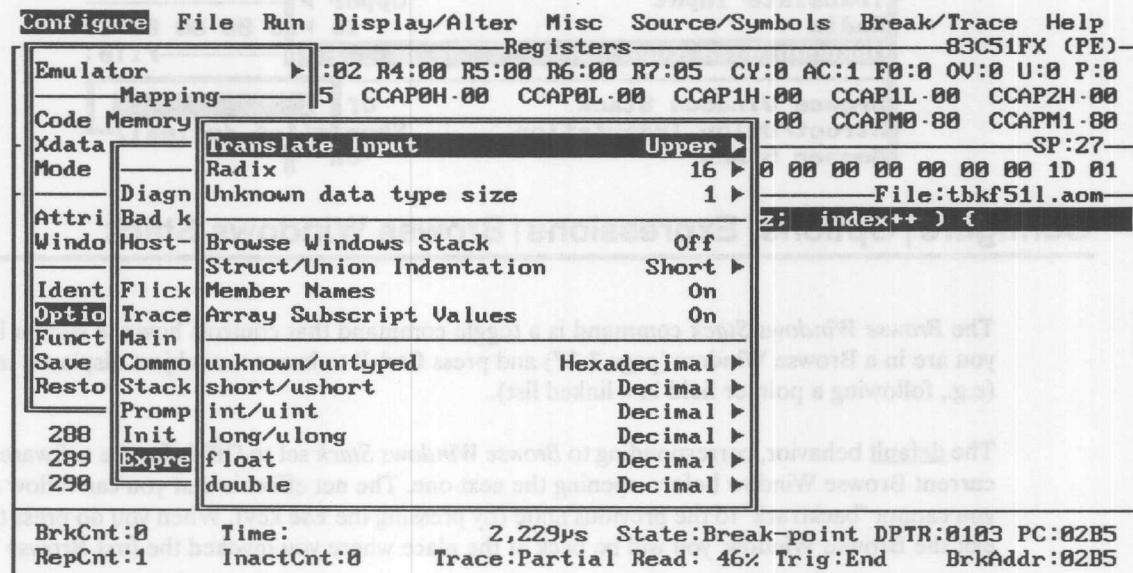
f()

... a function pointer to the function "f" is displayed as "f".

... a function pointer to the function "f" is displayed as "f".

Configure | Options | Expressions Command

The *Configure | Options | Expressions* command displays a pull-down menu of commands to allow you to control expression input (symbols and numbers), numeric value formatting, data structure formatting and data structure browsing features.

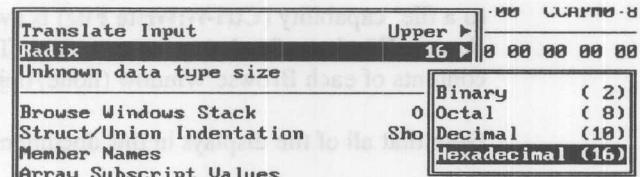


Case conversion to be applied to keyboard input and symbolic input from files

Configure | Options | Expressions | Radix

The *Radix* command allows you to specify the default radix (base) for numbers you enter that contain no explicit radix prefix or suffix. If you enter a number such as 10 or 101, it will be interpreted in the current default radix (base) as specified by this command. However, in certain contexts where it makes sense, the host software will temporarily override the current default. For example, when you enter a source line number as

#11



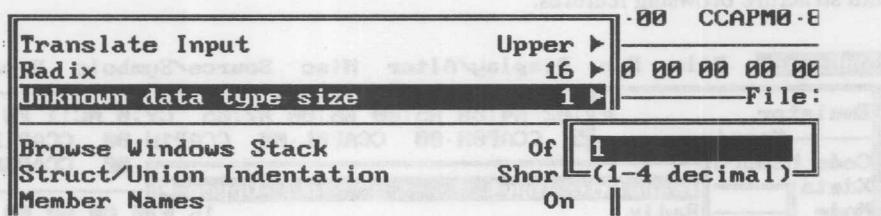
the software temporarily changes the default radix to 10 (decimal) before translating the '11'. In other cases, when the software displays a Dialog Box to obtain input, a prompt somewhere in the box (usually on the bottom border) indicates if the default radix has been changed temporarily.

In any case, you can always provide an explicit radix designator in the constant to override any default interpretation. For example, '#0x23' designates source line number 35 decimal, as does '#35t'.

In some cases, the software can automatically determine the appropriate radix based only on the characters and digits present in the constant. For example, 1A can only be the hexadecimal number 0x1A, even though no explicit prefix or suffix is present.

Configure | Options | Expressions | Unknown data type size

The *Unknown data type size* command tells the host software how many bytes to read, write or display for a value/symbol whose data type is "(unknown)" or "(untyped)".



Configure | Options | Expressions | Browse Windows Stack

The *Browse Windows Stack* command is a toggle command that controls how the system behaves when you are in a Browse Window (page 3-27) and press **Ctrl-B** to browse an object displayed in that window (e.g., following a pointer field in a linked list).

The default behavior, corresponding to *Browse Windows Stack* set to **Off**, is for the software to "close" the current Browse Window before opening the next one. The net effect is that you can follow a pointer, but you cannot "backtrack" to the previous node (by pressing the **Esc** key). When you do press the **Esc** key to exit the Browse Window, you will be back at the place where you invoked the first Browse Window.

A *Browse Windows Stack* setting of **On** causes successive pop-ups of a Browse Window to stack one on top of the other (recursively). You can backtrack to any of the previous Browse Windows by pressing the **Esc** key one or more times. As nice as it is, the reason that this is not the default system behavior is that you could cause the host software to overflow its own runtime stack by traversing too deeply into list or tree or whatever.

However, even when you have *Browse Windows Stack* set to **Off** (default setting), there is a way to at least maintain a record of the nodes/objects you have visited while browsing. The "write the data in this window to a file" capability (**Ctrl-W:Write File**) is available in a Browse Window, regardless of the setting of the *Browse Windows Stack* toggle command. Thus, at each step along the way, you can write the entire contents of each Browse Window (node/object), successively appending to the same file.

Note that all of the displays in this document were generated with *Browse Windows Stack* set to **On**.

Configure | Options | Expressions | Struct/Union Indentation

The *Struct/Union Indentation* command controls the indentation of structure and union members which are themselves structures, unions or arrays. The two indentation options available are '*Short*' and '*Long*'. The default is '*Short*' indentation.

With '*Short*' indentation, the members/elements of embedded structures, unions or arrays are indented two spaces from the beginning of the name of the "parent" member:

X:0164 ->struct Srt_Nd newnode			
Addr	Data Type	Value	
X:0104	struct Srt_Nd	struct Srt_Nd	
X:0104	union	lf_link	union
X:0104	->struct Srt_N	srx_left	X:0124
X:0104	->struct Srt_N	srx_fore	X:0124
X:0107	union	rb_link	union
X:0107	->struct Srt_N	srx_right	X:0144
X:0107	->struct Srt_N	srx_back	X:0144
X:010A	[4]char	srt_value	[4]char
X:010A	char	[0]	0x41 'A' 65
X:010B	char	[1]	0x42 'B' 66
X:010C	char	[2]	0x43 'C' 67
X:010D	char	[3]	0x00 '.' 0
X:010E	uchar	srt_vtype	0x04 '.' 4
X:010F	char	srt_char	0x4D 'M' 77
X:0110	int	srt_int	25
X:0112	uint	srt_flags	20
X:0114	long	srt_long	25
X:0118	ulong	srt_ulong	25
X:011C	float	srt_float	12345.678

With '*Long*' indentation, the members/elements of embedded structures, unions or arrays are indented two spaces from the beginning of the value field of the "parent" member:

X:0164 ->struct Srt_Nd newnode			
Addr	Data Type	Value	
X:0104	struct Srt_Nd	struct Srt_Nd	
X:0104	union	lf_link	union
X:0104	->struct Srt_N	srx_left	X:0124
X:0104	->struct Srt_N	srx_fore	X:0124
X:0107	union	rb_link	union
X:0107	->struct Srt_N	srx_right	X:0144
X:0107	->struct Srt_N	srx_back	X:0144
X:010A	[4]char	srt_value	[4]char
X:010A	char	[0]	0x41 'A' 65
X:010B	char	[1]	0x42 'B' 66
X:010C	char	[2]	0x43 'C' 67
X:010D	char	[3]	0x00 '.' 0
X:010E	uchar	srt_vtype	0x04 '.' 4
X:010F	char	srt_char	0x4D 'M' 77
X:0110	int	srt_int	25
X:0112	uint	srt_flags	20
X:0114	long	srt_long	25
X:0118	ulong	srt_ulong	25
X:011C	float	srt_float	12345.678

Note that this command has a visible effect only if the setting of the *Configure | Options | Expressions | Member Names* toggle command is 'On' (see page 3-18).

Configure | Options | Expressions | Member Names

The *Member Names* toggle command controls whether or not the host software will display structure and union member names when displaying the individual members of a structure or union. The default is 'On'.

This is an example of a union display generated with the *Member Names* toggle set to 'On' (the default):

Display/Alter Expression			
Expression: b_flags1_1			
Addr	Size	Type	
X:00FF	1 byte	Type: union B_Flags1	
		X:00FF union B_Flags1 b_flags1_1	
Addr	Data Type	Value	
X:00FF	union B_Flags1	union B_Flags1	
X:00FF	uchar	full_val 0x73 's' 115	
K_F	struct	b struct	
ags1_2.b	0:1 uchar	bf_1 0x01 '.' 1	
	1:1 uchar	bf_2 0x01 '.' 1	
Time:	2:1 uchar	bf_3 0x00 '.' 0	
InactCnt	3:3 uchar	bf_4 0x06 '.' 6	

Here is what the same display looks like with the *Member Names* toggle set to 'Off':

Display/Alter Expression			
Expression: b_flags1_1			
Addr	Size	Type	
X:00FF	1 byte	Type: union B_	
		X:00FF union B_Flags1 b_flags1_1	
Addr	Data Type	Value	
X:00FF	union B_Flags1	union B_Flags1	
X:00FF	uchar	0x73 's' 115	
K_F	struct	struct	
ags1_2.b	0:1 uchar	0x01 '.' 1	
	1:1 uchar	0x01 '.' 1	
Time:	2:1 uchar	0x00 '.' 0	
InactCnt	3:3 uchar	0x06 '.' 6	

Configure | Options | Expressions | Array Subscript Values

The *Array Subscript Values* toggle command controls whether or not the host software will display an array element's subscript value (enclosed in square brackets) before the actual value of the array element itself. The default is 'On'.

This is an example of an array display generated with the *Array Subscript Values* toggle set to 'On' (the default):

X:00FF [5]int scores		
Addr	Data Type	Value
X:00FF	[5]int	[5]int
X:00FF	int	[0] 0
X:0101	int	[1] 1
X:0103	int	[2] 4
X:0105	int	[3] 9
X:0107	int	[4] 16

Here is what the same display looks like with the *Array Subscript Values* toggle set to 'Off':

X:00FF [5]int scores		
Addr	Data Type	Value
X:00FF	[5]int	[5]int
X:00FF	int	0
X:0101	int	1
X:0103	int	4
X:0105	int	9
X:0107	int	16

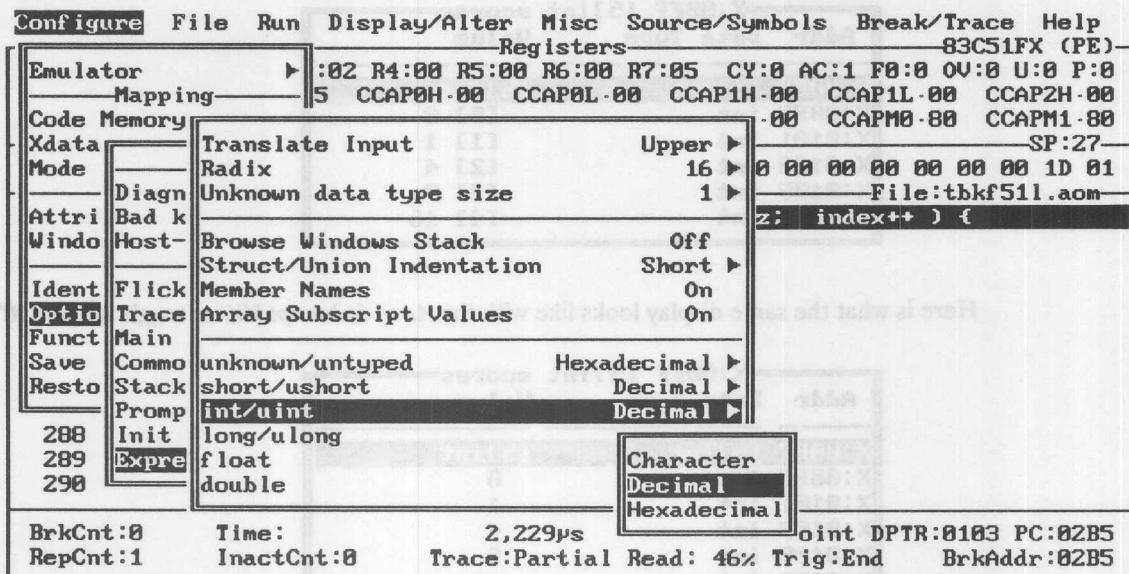
Configure Options Expressions Array Subscript Values	Configure Options Expressions Array Subscript Values	Configure Options Expressions Array Subscript Values
Configure	Configure	Configure
Options	Options	Options
Expressions	Expressions	Expressions
Array Subscript Values	Array Subscript Values	Array Subscript Values
On	Off	On
Help	Help	Help

Configure | Options | Expressions | Array Subscript Values

Configure | Options | Expressions | Array Subscript Values

Configure	Options	Expressions	unknown/untyped
Configure	Options	Expressions	short/ushort
Configure	Options	Expressions	int/uint
Configure	Options	Expressions	long/ulong
Configure	Options	Expressions	float
Configure	Options	Expressions	double

These commands control the display format to be used for values of the fundamental (basic) data types.



Display in decimal format

The available choices are:

Command	Controls Display Formatting for These Data Types	Display Formatting Options Available
Configure Options Expressions unknown/untyped	(unknown) & (untyped)	Character,
Configure Options Expressions short/ushort	short & unsigned short	Decimal
Configure Options Expressions int/uint	int & unsigned int	(signed or unsigned),
Configure Options Expressions long/ulong	long & unsigned long	or
Configure Options Expressions float	float	Hexadecimal
Configure Options Expressions double	double	

Note that there is no control needed for the **char** and **unsigned char** types, since values of these 1-byte types are always displayed in all three possible formats: hexadecimal, character and signed/unsigned decimal (in that order).

If you select Decimal format, signed or unsigned decimal notation will be used depending on the actual data type of the value (e.g., signed decimal for **int**, unsigned decimal for **unsigned int**).

Configure | Options | Expressions Command Combinations

The *Configure | Options | Expressions* commands operate, for the most part, independently of one another.

For example, the following structure display was generated with everything set to the "maximum" (*Struct/Union Indentation* = Long, *Member Names* = On, *Array Subscript Values* = On):

X:0164 ->struct Srt_Nd newnode			
Addr	Data Type	Value	
X:0104	struct Srt_Nd	struct Srt_Nd	
X:0104	union	lf_link	union
X:0104	->struct Srt_N		srx_left X:0124
X:0104	->struct Srt_N		srx_fore X:0124
X:0107	union	rb_link	union
X:0107	->struct Srt_N		srx_right X:0144
X:0107	->struct Srt_N		srx_back X:0144
X:010A	[4]char	srt_value	[4]char
X:010A	char	[0]	0x41 'A' 65
X:010B	char	[1]	0x42 'B' 66
X:010C	char	[2]	0x43 'C' 67
X:010D	char	[3]	0x00 '.' 0
X:010E	uchar	srt_vtype	0x04 '.' 4
X:010F	char	srt_char	0x4D 'M' 77
X:0110	int	srt_int	25
X:0112	uint	srt_flags	20
X:0114	long	srt_long	25
X:0118	ulong	srt_ulong	25
X:011C	float	srt_float	12345.678

Here is the same structure display generated with everything set to the "minimum" (*Struct/Union Indentation* = Short, *Member Names* = Off, *Array Subscript Values* = Off):

X:0164 ->struct Srt_Nd newnode			
Addr	Data Type	Value	
X:0104	struct Srt_Nd	struct Srt_Nd	
X:0104	union	union	
X:0104	->struct Srt_N	X:0124	
X:0104	->struct Srt_N	X:0124	
X:0107	union	union	
X:0107	->struct Srt_N	X:0144	
X:0107	->struct Srt_N	X:0144	
X:010A	[4]char	[4]char	
X:010A	char	0x41 'A'	65
X:010B	char	0x42 'B'	66
X:010C	char	0x43 'C'	67
X:010D	char	0x00 '.'	0
X:010E	uchar	0x04 '.'	4
X:010F	char	0x4D 'M'	77
X:0110	int	25	
X:0112	uint	20	
X:0114	long	25	
X:0118	ulong	25	
X:011C	float	12345.678	

Watch Window

Commands The *Configure|Windows|Watch* pull-down menu contains five commands to allow you to define and manipulate the content of the Watch Window.

The **Add** command lets you add any non-constant expression to the Watch Window. Note that the host software reevaluates the address designated by the watch expression each time the Watch Window is updated. Thus, if you add the expression *my_array[i]* to the Watch Window, the host software recomputes the subscript value of *i* each time the software updates or repaints the Watch Window.

The **Delete** command deletes (removes) a single, designated expression from the Watch Window.

The **Clear** command deletes (removes) all watch expressions from the Watch Window.

The **Save** command saves (writes) all the watch expressions in the Watch Window to a file of your choice. You can later load (add) these saved watch expressions to the Watch Window by selecting the *Configure|Windows|Watch|Load* command below.

The **Load** command loads (restores) into the Watch Window the watch expressions saved in a file by the *Configure|Windows|Watch|Save* command above.

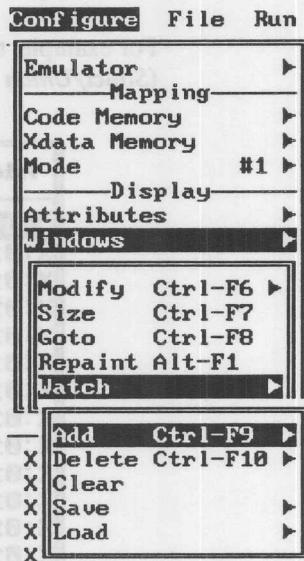
The various commands, or the entire menu, may be inactive (inaccessible) depending on the current state of the system. For example, if the Watch Window is empty, the Delete, Clear and Save commands will be inactive (inaccessible).

Display Format

The Watch Window has four columns, showing the address of the watch expression, the data type of the watch expression, the watch expression itself and the current value of the watch expression:

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help			
Module: TA	Source	File: takf511.asm	
76	aj = (ai * 2) + main_loop_factor;		
77	ary[aj] = aj;		
		Watch	
X:008E	uint	ai	6
X:008A	uchar	ary[aj]	0x2F '7' 47
X:0084	[10]uchar	ary	[10]uchar
X:0084	uchar		[0] 0x24 '\$' 36
X:0085	uchar		[1] 0x26 '&' 38
X:0086	uchar		[2] 0x28 '(' 40
X:0087	uchar		[3] 0x2A '*' 42
X:0088	uchar		[4] 0x2C ',' 44
X:0089	uchar		[5] 0x2E '.' 46
X:008A	uchar		[6] 0x2F '/' 47
X:008B	uchar		[7] 0x31 '1' 49
X:008C	uchar		[8] 0x33 '3' 51
X:008D	uchar		[9] 0x35 '5' 53
X:00A8	->struct List	p_list	(null)
C:0000?struct List		*p_list	struct List
C:0000?uint			list_uival 517
C:0002?->struct List			list_p_nxt ?:FFFF{CEFFFF}
X:0080	float	iteration	38.5

Begin emulation



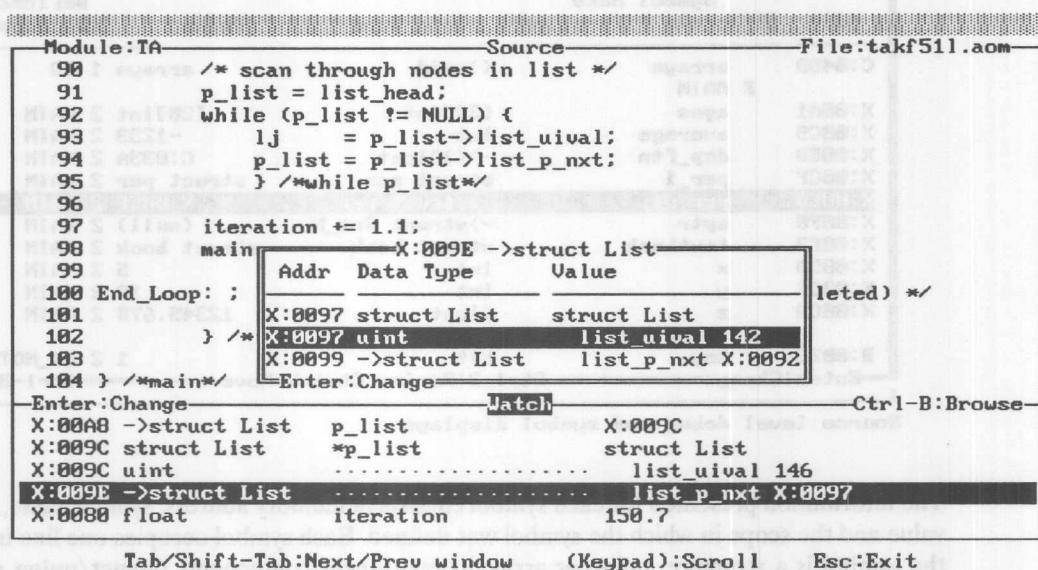
If the watch expression is a structure, union or array type, then there will also be a line for each constituent component in the watch expression: each member in a **struct/union** or each element in an array. In this case, the watch expression field contains ".....".

Every time the host software updates or repaints the Watch Window, it highlights the value field of each watch expression or constituent component if the current value is different from the previous value. This highlighting is done to notify you of the change in value. Similarly, if the address associated with the watch expression or constituent component changes, the software highlights the address field. (The address associated with a watch expression can change if you are watching an expression such as *my_array[i]*, where the actual array element being monitored changes as the value of *i* changes.)

If the host software encounters an "illegal" value in evaluating the address of a watch expression, it flags the watched object with a question mark (?) between the address column and the data type column. This can happen if the watch expression involves indirection through a pointer and the value of that pointer is NULL or illegal (see page 3-13).

Browsing

You can enter Change Mode to browse the Watch Window by pressing the **Tab** key (which is simply a convenient shortcut for the **Configure|Windows|Goto** command) one or more times until the Watch Window title is highlighted:



Addr	Data Type	Value
X:0097	struct List	struct List
X:0097	uint	list_uival 142
X:0099	->struct List	list_p_nxt X:0092

Module:TA Source File:takf511.aom

Enter:Change Watch Ctrl-B:Browse

Tab/Shift-Tab:Next/Prev window (Keypad):Scroll Esc:Exit

As you scroll up or down in the Watch Window, the software will display prompts on the top border of the Watch Window indicating what actions you can perform on the highlighted item:

Enter:Change – (upper-left border) The currently highlighted item is a fundamental value or a pointer, either of which you can change. Pressing the **Enter** key pops up a Change Box (see page 3-26), allowing you to change the item's current value.

Ctrl-B:Browse – (upper-right border) The currently highlighted item is a pointer which can be "followed". Pressing the **Ctrl-B** key sequence pops up a Browse Window (see page 3-27), allowing you to browse/inspect the object pointed to by the pointer value.

Source/Symbols Displays

The following *Source/Symbols* windows display the internal symbol table sorted in various ways:

Source/Symbols | Global: global (PUBLIC) symbols, sorted alphabetically

Source/Symbols | Local: local symbols, sorted alphabetically within each scope

Source/Symbols | Alpha: global and local symbols, sorted alphabetically

Source/Symbols | Address: global and local symbols, sorted by address

All these windows have the same basic display format. Following is a display generated by the *Source/Symbols | Local* command:

Enter Symbol	Local Symbols - By Module					(104 Symbols)
	Symbol Name	Addr	(M:Module F:Function)	Data Type	Value	Defined Within (#: Scope Level)
C:04DD	arrays			void		arrays 1 TB
	F MAIN					
X:00A1	ages			[20]int	[20]int	2 MAIN
X:00C9	average			int	-1233	2 MAIN
X:00ED	dsp_ftn			->()float	C:033A	2 MAIN
X:00CF	per_1			struct per	struct per	2 MAIN
X:00EA	per_p1			->struct per	X:00CF	2 MAIN
X:00F0	sprt			->struct Srt_M	(null)	2 MAIN
X:00F3	textbook			struct book	struct book	2 MAIN
X:009D	x			int	5	2 MAIN
X:009F	y			int	10	2 MAIN
X:00CB	z			float	12345.678	2 MAIN
	F DO NOTHING					
B:0028	key			bit		1 2 DO NOTHING

Source level debug and symbol displays

The information presented for each symbol consists of memory address, symbol name, data type, current value and the scope in which the symbol was defined. Each symbol occupies one line in these displays. If the symbol is a structure, union or array, its constituent components (struct/union members or array elements) are not shown in these displays. To see the individual struct/union members or array elements, you must press the "browse" key combination (Ctrl-B:Browse – see below).

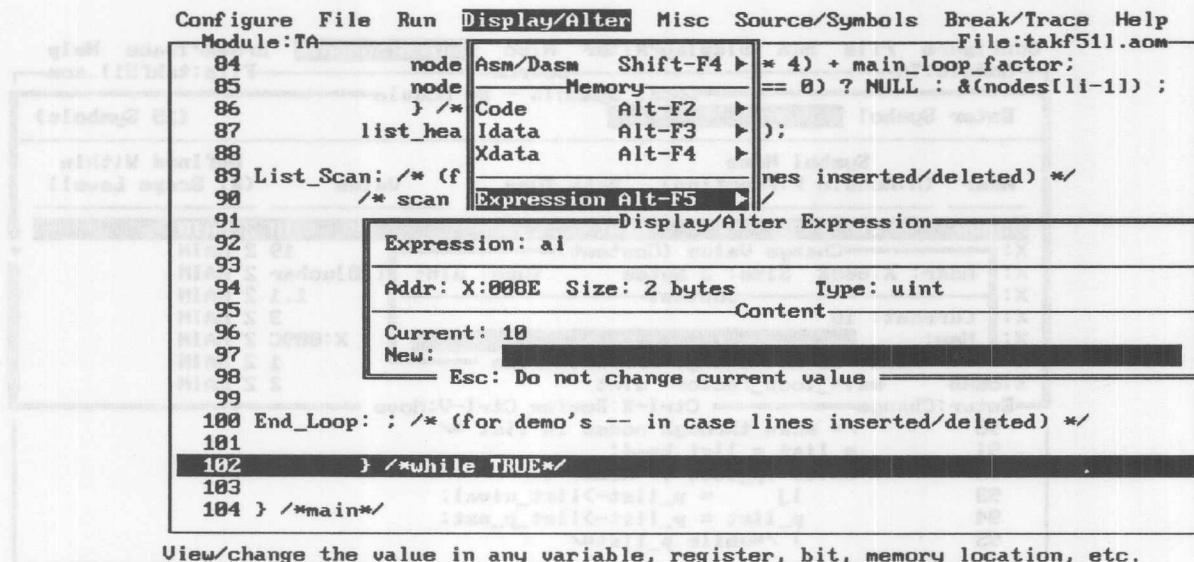
As you scroll up or down in these *Source/Symbols* windows, the software will display prompts on the bottom border of the window indicating what actions you can perform on the highlighted symbol:

Enter:Change – (bottom-left border) The currently highlighted symbol's data type is one of the fundamental types or it is a pointer. Pressing the **Enter** key pops up a Change Box (see page 3-26), allowing you to change the symbol's current value.

Ctrl-B:Browse – (bottom-right border) The currently highlighted symbol is an array, structure or union, or it is a pointer (which can be "followed"). Pressing the **Ctrl-B** key sequence pops up a Browse Window (see page 3-27), allowing you to browse/inspect the array elements, struct/union members or the object pointed to by the pointer variable.

Display/Alter | Expression

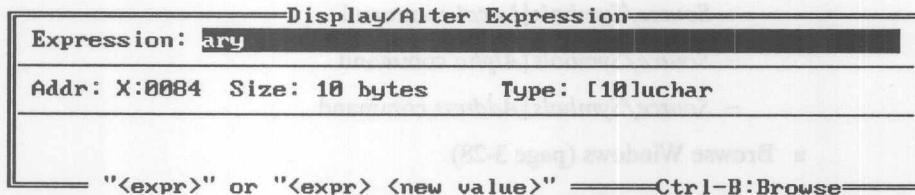
The *Display/Alter | Expression* command pops up a dialog box allowing you to view or change the value stored in a location designated by an arbitrary expression.



View/change the value in any variable, register, bit, memory location, etc.

The dialog box shows the address, size (in bytes or bits) and data type of the expression you enter. If the expression designates a location whose content can be changed, the "Current:" field shows the value currently stored in that location.

If the expression you enter designates a structure, union or array value, the software shows the address, size and data type of the expression. Additionally, the **Ctrl-B:Browse** prompt appears on the bottom-right border of the box to indicate that you can pop up a Browse Window (page 3-27) to inspect and/or change the individual members in the **struct/union** or the individual elements in the array:

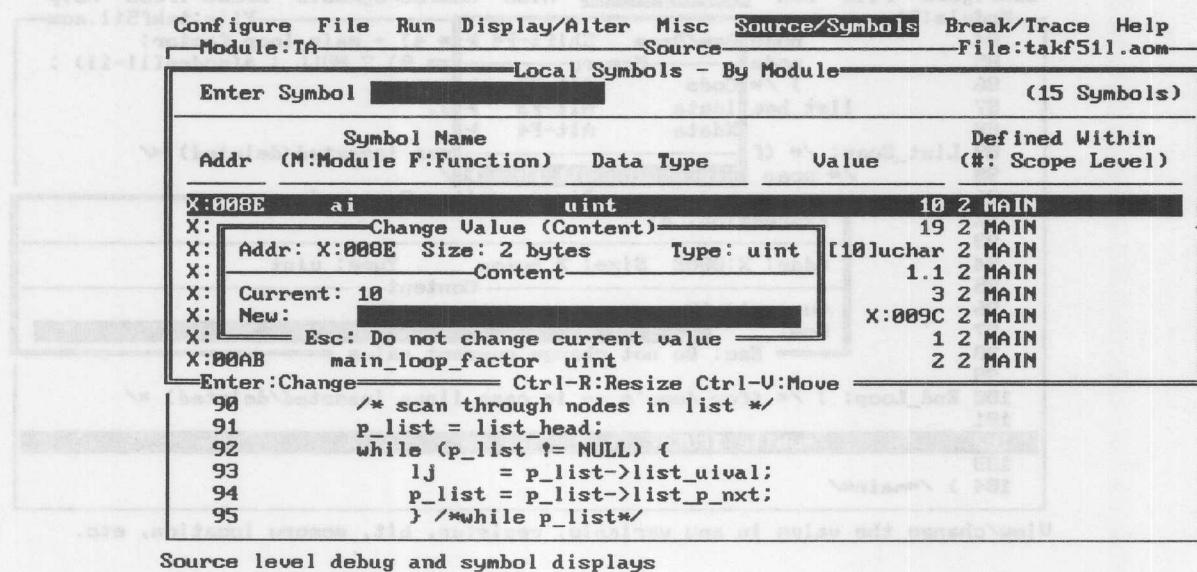


If the expression you enter designates a "pointer to..." value, you have the option of either changing the pointer, by entering a new value, or "following" the pointer, by pressing **Ctrl-B** (Browse).

Note: Currently, the expression entered in the "New:" field is evaluated for address, not for value. For example, if you enter *i*, the address of the variable *i* is used as the new value, not the content of the variable *i*.

Change Box

A Change Box is very much like the *Display/Alter|Expression* command's pop-up dialog box. The difference is that, with a Change Box, the expression or variable whose value is to be changed is already known.



A Change Box pops up when you press the **Enter** key in response to the **Enter:Change** prompt in the following contexts:

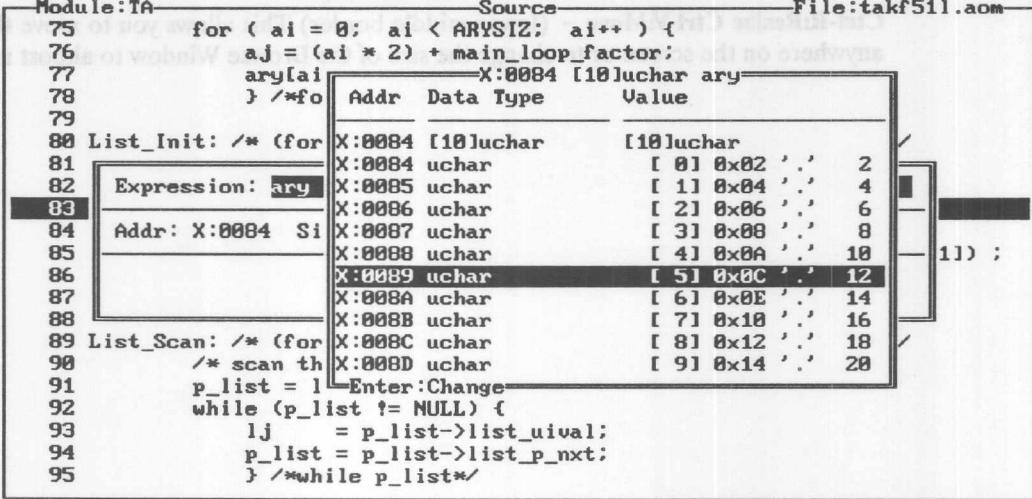
- Watch Window (page 3-23)
- Source/Symbols windows (page 3-24)
 - *Source/Symbols|Global* command
 - *Source/Symbols|Local* command
 - *Source/Symbols|Alpha* command
 - *Source/Symbols|Address* command
- Browse Windows (page 3-28)

Browse Windows

A Browse Window pops up to allow you to view (inspect) and/or change the

- members in a structure or union
- elements in an array
- object designated by a pointer value (i.e., to "follow" the pointer)

Here is a Browse Window popped-up from the *Display/Alter|Expression* command to examine the individual elements in the array *ary*:



The screenshot shows a 'Display/Alter' window with the following details:

- Module: TA**
- File: takf511.aom**
- Source:** The code is displayed in the source pane:

```
75     for ( ai = 0; ai < ARYSIZ; ai++ ) {  
76         aj = ( ai * 2 ) + main_loop_factor;  
77         ary[ai] = X:0084 [10]uchar ary  
78     } /*for  
79  
80 List_Init: /* (for  
81  
82     Expression: ary  
83     Addr: X:0084 Si  
84  
85     X:0084 [10]uchar  
86     X:0084 uchar [ 0] 0x02 . . . 2  
87     X:0085 uchar [ 1] 0x04 . . . 4  
88     X:0086 uchar [ 2] 0x06 . . . 6  
89     X:0087 uchar [ 3] 0x08 . . . 8  
90     X:0088 uchar [ 4] 0x0A . . . 10  
91     X:0089 uchar [ 5] 0x0C . . . 12  
92     X:008A uchar [ 6] 0x0E . . . 14  
93     X:008B uchar [ 7] 0x10 . . . 16  
94     X:008C uchar [ 8] 0x12 . . . 18  
95     X:008D uchar [ 9] 0x14 . . . 20  
96     /* scan th  
97     p_list = 1 Enter:Change  
98     while ( p_list != NULL ) {  
99         l_j = p_list->list_uival;  
100        p_list = p_list->list_p_nxt;  
101    } /*while p_list*/
```

- Display/Alter memory:** The memory dump pane shows the array elements:

Addr	Data Type	Value
X:0084	[10]uchar	[0] 0x02 . . . 2
X:0084	uchar	[1] 0x04 . . . 4
X:0085	uchar	[2] 0x06 . . . 6
X:0086	uchar	[3] 0x08 . . . 8
X:0087	uchar	[4] 0x0A . . . 10
X:0088	uchar	[5] 0x0C . . . 12
X:0089	uchar	[6] 0x0E . . . 14
X:008A	uchar	[7] 0x10 . . . 16
X:008B	uchar	[8] 0x12 . . . 18
X:008C	uchar	[9] 0x14 . . . 20

A Browse Window pops up when you press the **Ctrl-B** key in response to the **Ctrl-B:Browse** prompt in the following contexts:

- Watch Window (page 3-23)
- Source/Symbols windows (page 3-24)
 - *Source/Symbols|Global* command
 - *Source/Symbols|Local* command
 - *Source/Symbols|Alpha* command
 - *Source/Symbols|Address* command
- *Display/Alter|Expression* command (page 3-25)
- Browse Windows (e.g., to follow a pointer which is a **struct/union** member or an array element)

Structures, unions and arrays are fully "expanded" in a Browse Window to show all constituent components: each member in a **struct/union** or each element in an array. This process applies iteratively if a component is itself a structure, union or array.

The *Configure|Options|Expressions|Browse Window Stack* toggle command (page 3-16) controls how the system behaves when you are in a Browse Window and press **Ctrl-B** to browse an object displayed in that window (e.g., following a pointer field in a linked list).

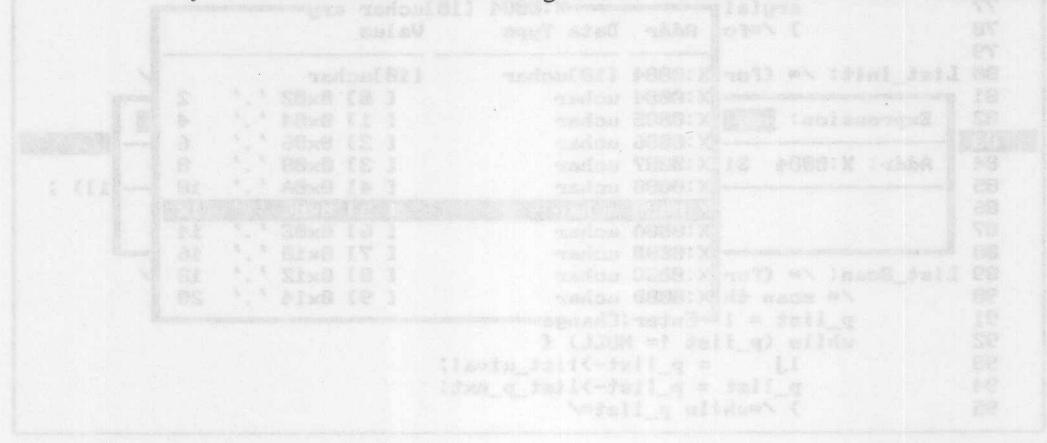
The following keys are available in a Browse Window:

Enter:Change – (lower-left border) The currently highlighted item is a fundamental value or a pointer, either of which you can change. Pressing the **Enter** key pops up a Change Box (see page 3-26), allowing you to change the item's current value.

Ctrl-B:Browse – (lower-right border) The currently highlighted item is a pointer which can be "followed". Pressing the **Ctrl-B** key sequence pops up another Browse Window, allowing you to browse/inspect the object pointed to by the pointer value.

Ctrl-W:Write – (upper-right border) Pressing **Ctrl-W** allows you to write the entire contents of the Browse Window to a file of your choice. The software will prompt you for the name of the file and ask you whether you want to append to that file (add to the end of the file) or overwrite it (start over at the beginning of the file).

Ctrl-R:Resize Ctrl-V:Move – (lower-middle border) This allows you to move the Browse Window anywhere on the screen or to change the size of the Browse Window to almost any size you want.



Compilers

This section describes how to get the most out of the capabilities in the *iceMASTER* host software in combination with the particular compiler product you are using. *iceMASTER* currently provides full data type support (structures, unions, arrays, etc.) for the compilers listed in this section. More comprehensive support for other compilers will be added in the near future.

Keil/Franklin C51

This information applies to version V3.07 of C51 (C compiler) and version V2.7 of L51 (linker). It should be accurate for later versions of these products and may even be valid for earlier versions.

Recommended Compilation Options

DEBUG	Include debug information in the object file.
OBJECTEXTEND	Include full data type definition debug information in the object file.
NOREGPARMS	Do not pass function parameters in registers.
OPTIMIZE(3)	Do not put auto variables in registers.

Recommended Linking Options

DEBUGLINES	Include source line number debug information in the output file.
DEBUGPUBLICS	Include public (global) symbol information in the output file.
DEBUGSYMBOLS	Include all symbol information in the output file.

Pointers

The Keil/Franklin C51 compiler implements both generic pointers and memory-specific pointers (see page 3-13).

NULL

The value of a NULL generic pointer is a pointer value with all three bytes containing zero (even the selector byte). This is true regardless of whether the pointer is being read (e.g., for comparison to NULL) or written (i.e., NULL being assigned to the pointer).

'typedef' Names

The *iceMASTER* host software is internally set up to display **typedef** names, where appropriate. However, the compiler does not output any information about **typedefs**. The compiler always outputs the underlying type. For example, in

```
typedef struct abc {...} ABC;
ABC xyz;
```

There is no information about 'ABC' in the AOM file (Absolute Object Module file, load module file). The data type of 'xyz' is recorded as '**struct abc**', not '**typedef ABC**'. Likewise, for

```
typedef unsigned char logical;
logical bool_val;
```

there is no information about 'logical' and the type of 'bool_val' is recorded as **unsigned char**.

Enumeration Types

The *iceMASTER* host software is internally set up to display **enumeration** values as they were declared in the C source program (e.g., 'RED', 'GREEN', 'YELLOW', rather than 0, 1, 2). However, the compiler does not output **enumeration** type information. The AOMF-specified type of **enum** variables always seems to be **signed int**.

'double' Types

The data type **double** (double-precision floating-point) is implemented by the compiler as **float** (single-precision floating-point). The data type information in the AOM file for variables declared as 'double' reflects this.

Batch "Make" Files

The batch "make" files TAKF51.BAT and TBKF51.BAT show how the example programs TA.C and TB.C were compiled and linked.

This information applies to version V4.23E/DXT of C-51 (C compiler) and version V4.44D/DXT of XLINK (linker). It should be accurate for later versions of these products and may even be valid for earlier versions.

Recommended Compilation Options

-r0n Put only symbolic information (no source code) into the object file.
Do not add extra NOPs at source line boundaries.

Do not specify '-r1...' or '-r2...': The *iceMASTER* emulators break emulation before the instruction at a break-point is executed. Thus, no extra NOPs are needed in the generated code.

Warning: Do not specify '-r0i...' (add #include file source code to object file) and do not forget to specify the 'n' in '-r0n'. The host software ignores the source images embedded in the load module file. *iceMASTER* locates your original source file on disk and reads that file when displaying source images. Additionally, the line numbers recorded in the resultant load module file (for '-r0i...') will not match the original source file found on your disk.

Recommended Linking Options

-Fdebug Create the load module file in the IAR/Archimedes DEBUG format (.DBG).

Pointers

The IAR/Archimedes C-51 compiler implements only generic pointers; there are no memory-specific pointers (see page 3-13).

NULL

The value of a NULL generic pointer is dependent on the context.

Reading	In comparing a pointer value to NULL, only the offset portion (<i>hhhh</i>) is examined (for both bytes being zero). The value in the memory space selector byte is ignored.
Writing	When NULL (zero) is assigned to a pointer, all three bytes in the pointer (including the memory space selector byte) are set to zero.

'typedef' Names

The *iceMASTER* host software is internally set up to display **typedef** names, where appropriate. However, the compiler does not output any information about **typedefs**. The compiler always outputs the underlying type. For example, in

```
typedef struct abc {...} ABC;
ABC xyz;
```

There is no information about 'ABC' in the .DBG file (load module file). The data type of 'xyz' is recorded as 'struct abc', not '**typedef ABC**'. Likewise, for

```
typedef unsigned char logical;
logical bool_val;
```

there is no information about 'logical' and the type of 'bool_val' is recorded as **unsigned char**.

Enumeration Types

The *iceMASTER* host software is internally set up to display **enumeration** values as they were declared in the C source program (e.g., 'RED', 'GREEN', 'YELLOW', rather than 0, 1, 2). However, the compiler does not output **enumeration** type information. The .DBG-specified type of **enum** variables always seems to be **signed int**.

'double' Types

The data type **double** (double-precision floating-point) is implemented by the compiler as **float** (single-precision floating-point). The data type information in the .DBG file for variables declared as 'double' reflects this.

Batch "Make" Files

The batch "make" files TAIA51.BAT and TBIA51.BAT show how the example programs TA.C and TB.C were compiled and linked.

Appendix A: Source Window & Assembler/Disassembler

Source Window

The **Source Window** is used to display the source program and/or disassembled instructions in Code Memory. The current module (see the *Source/Symbols | Current Module* command) is displayed on the left side of the top border and the filename of the currently loaded program (see the *File | Load* command) is displayed on the right side of the top border.

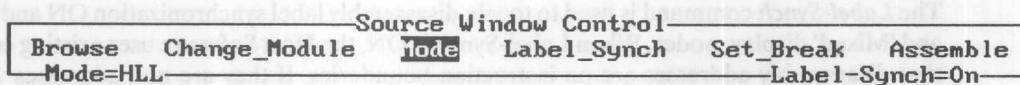
The information in this window may be displayed in 'Code', 'Mixed' or 'HLL' mode. 'Code' means just an assembly language disassembly of the code will be displayed. 'Mixed' means that HLL source lines (e.g., C language), if available, will be displayed along with the disassembled code. 'HLL' means that just the HLL source lines will be displayed. The default is 'HLL' but can be changed under the *Options* column of the *Configure | Windows | Modify* command.

The starting address of the code displayed is at the PC address and cannot be changed. Note that during single step (*Run | Step*) or slow motion operation (*Run | Slow Motion*) the starting address of the screen does not change until the flow of execution moves out of the code that appears in the window, at which time a new window full of code will be displayed. This is intended to keep the Dynamically Annotated Code visible as long as possible.

In Change Mode, the Source Window also allows you to control the emulation environment.

You can enter Change Mode to browse/peruse the Source Window by pressing the **Tab** key (which is simply a convenient shortcut for the *Configure | Windows | Goto* command) one or more times until the Source Window title is highlighted.

Once you are in Change Mode, by pressing **←**, **→** or **Enter**, the *Source Window Control* Menu becomes available.



Source Window Control

The *Source Window Control* Menu is a menu bar window which allows you to browse through or change your code, change the display mode, turn disassembly label synchronization ON and OFF, set or clear simple break-points and set temporary break-points. The commands are:

Browse

The *Browse* command allows you to quickly reposition the highlight bar in the Source Window. If the Source Window display mode is *Code* or *Mixed*, you will be prompted for an address to go to. If the Source Window display mode is *HLL* (source only), you will be prompted for a line number. However, you may enter any address expression (numerically or symbolically) or line number at either prompt. Note that if the display mode is *HLL* and you want to switch to a different module, use the *Change Module* command.

Change Module

In *HLL* (source-only) or *Mixed* display mode, you can use the *Change Module* command to display (switch to) a different module in the Source Window.

Mode

The *Mode* command calls a Pull-down Menu from which you may set the display mode for the Source Window. The available choices are *Code*, *Mixed* and *HLL* (source only).

Mode | Code

The *Code* command changes the display mode of the Source Window to a disassembled view of code memory.

Mode | Mixed

The *Mixed* command changes the display mode of the Source Window to a disassembled view of code memory, interspersed with HLL source line images for code-generating source lines.

Mode | HLL

The *HLL* command changes the display mode of the Source Window to a source-only view of the HLL source module corresponding to the currently highlighted line. This display mode shows both code-generating and non-code-generating source lines.

Label-Synch

The *Label-Synch* command is used to toggle disassembly label synchronization ON and OFF in the 'Code' and 'Mixed' display modes. When Label-Synch is ON, the Host Software uses existing code labels to verify that disassembly addresses are on instruction boundaries. If they are not, addresses will be adjusted so that the disassembly does begin on an instruction boundary. If Label-Synch is OFF, no such checks are made and the disassembly will start at the specified address.

Set Break

The *Set Break* command calls a Pull-down Menu from which you may set or clear simple break-points and set a temporary break-point.

Set Break | Toggle

The *Toggle* command toggles (on or off) a permanent simple break-point at the currently highlighted line. Permanent simple break-point locations are flagged with the \triangleright character and may be modified in the *Break/Trace | Set* menu.

Note that **Ctrl-B** (toggle break-point) performs the same function as the *Set Break | Toggle* command.

Set Break | Run Until

The *Run Until* command sets a temporary break-point at the currently highlighted line and then begins emulation immediately from the current PC. This command is essentially the same as the primary *Run | Until* command, except that the temporary break-point address is implicitly denoted as the currently highlighted line in the Source Window.

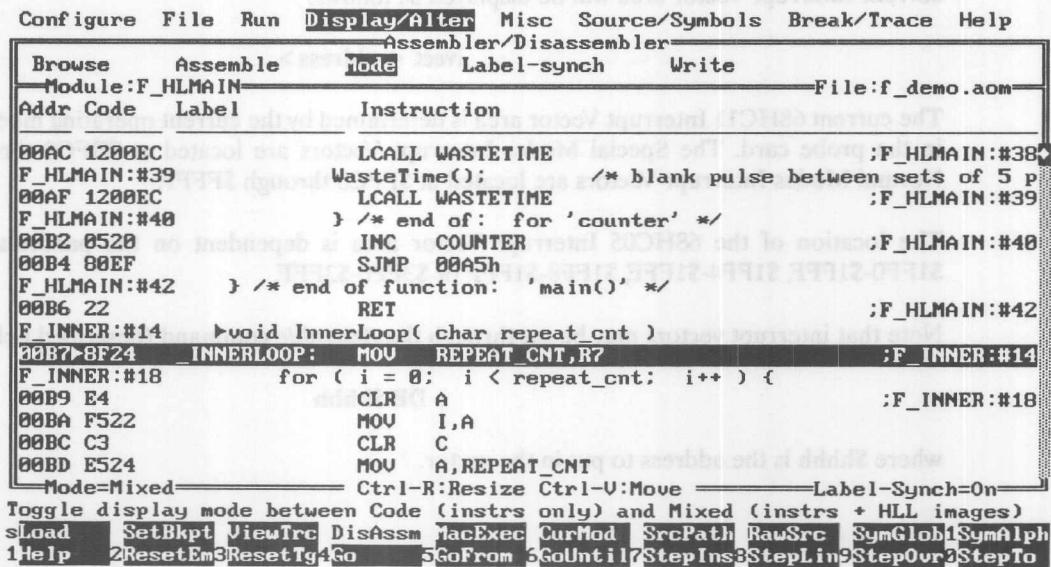
Note that this "set temporary break-point and start emulation" function may also be accomplished directly from the source window by pressing **Ctrl-T** (set temporary break-point and go).

Assemble

The *Assemble* command invokes the *Assembler/Disassembler* menu (*Display/Alter | Asm/Dasm* command) to allow you to browse through or change (patch) code.

Display/Alter | Asm/Dasm

The *Display/Alter | Asm/Dasm* command calls Assembler/Disassembler Menu Bar Window from which you may assemble instructions into code memory and browse through the disassembled code memory. Browse mode is active on entrance to the *Asm/Dasm* Menu.



The screenshot shows the Assembler/Disassembler menu bar window. The menu bar includes: Configure, File, Run, Display/Alter, Misc, Source/Symbols, Break/Trace, Help, Assembler/Disassembler, and a status line showing 'File:f_demo.aom'. The window has a toolbar with buttons for Browse, Assemble, Mode, Label-synch, and Write. The status bar at the bottom shows 'Mode=Mixed', 'Ctrl-R:Resize Ctrl-U:Move', 'Label-Synch=On', and a toggle for display mode between Code (instrs only) and Mixed (instrs + HLL images). The main area displays assembly code for 'F_HLMAIN' in 'f_demo.aom'. The code includes instructions like LCALL, INC, SJMP, RET, and MOV, along with comments and labels. The assembly window has scroll bars and a status bar at the bottom with various assembly-related commands.

Note that the right side of the top border line (under the Menu Bar) shows the filename of any loaded code file, the left side of the bottom border line shows the Display Mode, the center of the bottom border line shows current key information and the right side of the bottom border line shows the Label-synch mode. The commands are:

Display/Alter | Asm/Dasm | Browse

The *Browse* command calls a dialog box from which you may enter an address expression. If an address is entered the display will be repainted starting at the specified address.

Note that if an absolute address is entered and the Label-synch Mode is OFF, the address may or may not correspond to an instruction boundary and may result in an 'out-of-synch' disassembly.

Note for iceMASTER-68HC11 and iceMASTER-68HC05 disassemblies: Several opcode mnemonics have aliases (alternative mnemonic names). In such cases, the alternative name will be shown as a comment next to the disassembled instruction:

68HC11		68HC05	
Mnemonic	Alias	Mnemonic	Alias
ASL	LSL	BCC	BHS
ASLA	LSLA	BCS	BLO
ASLB	LSLB	DECX	DEX
ASLD	LSLD	INCX	INX
BCC	BHS	LSL	ASL
BCS	BLO	LSLA	ASLA
		LSLX	ASLX

You can use either form shown above when patching in new instructions using the single-line assembler (*Display/Alter|Asm/Dasm|Assemble* command).

In addition, for the iceMASTER-68HC11 and iceMASTER-68HC05, disassembled code memory at the current Interrupt Vector area will be displayed as follows:

ivect <address>

The current 68HC11 Interrupt Vector area is determined by the current operating mode of the processor in the probe card. The Special Modes Interrupt Vectors are located at \$BFC0 through \$BFFF. The Normal Modes Interrupt Vectors are located at \$FFC0 through \$FFFF.

The location of the 68HC05 Interrupt Vector area is dependent on the particular chip derivative: \$1FF0-\$1FFF, \$1FF4-\$1FFF, \$1FF8-\$1FFF or \$3FF0-\$3FFF.

Note that interrupt vectors may be set through the *Assemble* command (described below) using

DB \$hhhh

where \$hhhh is the address to put in the vector.

Assemble | **memD\memA** | **retlA\valc**

scratches are H. noisestype scratches are visible when using display mode and go to a file > background scratch edT
scratches background out in graphics bottleneck ed file valc is visible

Display/Alter | Asm/Dasm | Assemble

The *Assemble* command turns on Assemble Mode. In this mode you may enter new instructions (and labels) into code memory a single line at a time, at the current (highlighted) address. The address at which to add the instruction may be changed by using *Browse* mode (described above) or by using the ↓ or ↑ keys to increment or decrement the address.

```
Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help
Assembler/Disassembler
Browse Assemble Mode Label-synch Write
Module:F_H New Instruction at 00B7 File:f_demo.aom
Addr Code mov repeat_cnt,5

00AC 1200EC LCALL WASTETIME :F_HLM MAIN:#38
F_HLM MAIN:#39 WasteTime(); /* blank pulse between sets of 5 p :F_HLM MAIN:#39
00AF 1200EC LCALL WASTETIME
F_HLM MAIN:#40 } /* end of: for 'counter' */
00B2 0520 INC COUNTER :F_HLM MAIN:#40
00B4 00EF SJMP 00A5h
F_HLM MAIN:#42 } /* end of function: 'main()' */
00B6 22 RET :F_HLM MAIN:#42
F_INNER:#14 void InnerLoop( char repeat_cnt )
00B7>8F24 INNERLOOP: MOU REPEAT_CNT,R7 :F_INNER:#14
F_INNER:#18 for ( i = 0; i < repeat_cnt; i++ ) {
00B9 E4 CLR A :F_INNER:#18
00BA F522 MOU I,A
00BC C3 CLR C
00BD E524 MOU A,REPEAT_CNT
Mode=Mixed Ctrl-R:Resize Ctrl-U:Move Label-Synch=On
Assemble instructions into code memory (patch code memory with new instructions)
sLoad SetBkpt ViewTrc DisAssm MacExec CurMod SrcPath RawSrc SymGlob1SymAlph
1Help 2ResetEm3ResetTg4Go 5GoFrom 6GoUntil7StepIns3StepLin9StepOvr2StepTo
```

When entering an instruction, symbolic information may be used. This includes the ability to define a new label at the current address and to use symbolic names in the operand part of the instruction. When a new label is entered, it is inserted into the internal symbol table as a global (public) symbol.

Note that if the assembled instruction does not contain the same number of bytes as the original instruction at the current address you will be notified and asked to verify the operation through a Confirmation Box. Note that the *Assemble* command is enabled only for a line of machine code, *NOT* for HLL source statements.

To exit Assemble Mode press Esc to return to the Menu Bar.

Display/Alter | Asm/Dasm | Mode

The *Mode* command is used to toggle the display mode between 'Code' Mode and 'Mixed' Mode. The 'Code' display Mode means that just assembly language instructions are displayed. The 'Mixed' display Mode (available only if HLL source is available) means that HLL source images are interspersed with the assembly code.

Display/Alter | Asm/Dasm | Label-synch

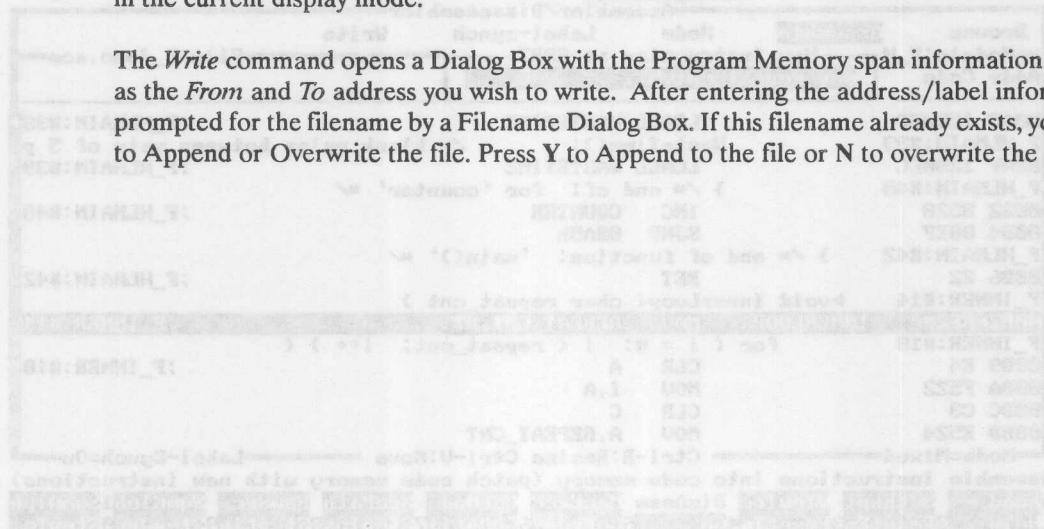
The *Label-synch* command is used to toggle disassembly label synchronization ON and OFF. When Label-synch is ON, the Host Software uses existing code labels to verify that disassembly addresses are on instruction boundaries. If they are not the Host Software will adjust the address so the disassembly begins on an instruction boundary. If the Label-synch mode is OFF, no such checks are made and the disassembly will start at the specified address.

Note that if Label-synch Mode is OFF, the specified address may or may not correspond to an instruction boundary and may result in an 'out-of-synch' disassembly.

Display/Alter | Asm/Dasm | Write

The *Write* command is used to write the disassembled contents of code memory to a disk file. The format of the information written to that file will be just as you see it on the screen (i.e., in human-readable form), in the current display mode.

The **Write** command opens a Dialog Box with the Program Memory span information given to you as well as the *From* and *To* address you wish to write. After entering the address/label information, you will be prompted for the filename by a Filename Dialog Box. If this filename already exists, you will be requested to Append or Overwrite the file. Press Y to Append to the file or N to overwrite the file.



Appendix B: Virtual Memory Support

Command Line Options

```
-vme <pages_per_allocation_request>
-vmf <num_bufs> <buf_size>
-vmfn <file_name>
```

These options control how virtual heap memory will be used. Enabling virtual heap memory allows the Host Software to use memory located outside the conventional 0-640Kb memory area normally usable by a DOS program. This capability is useful if you have a program with a large number of symbols which cannot all fit into the symbol table in conventional memory.

By default, the virtual heap memory is disabled and the Host Software builds the symbol table in conventional memory when loading a program file. If you ever see the following message at the top of the screen

"Not enough unused memory available to continue ... press any key."

the Host Software has used up all the conventional memory and, generally, cannot continue. If this happens, you should reinvoke the Host Software, specifying one or more of the command line options (-vme, -vmf and -vmfn) to enable virtual heap memory usage. However, note that loading files (building the internal symbol table) and some other commands within the Host Software (e.g., listing all symbols alphabetically) may be much slower when virtual heap memory is enabled.

There are two basic kinds of backing store media (real memory/storage) used to support the virtual heap memory:

- 1) LIM EMS 3.2/4.0 Expanded Memory, and
- 2) a file

The fastest medium is expanded memory and you should use this form if:

- 1) You have an expanded memory card in your computer (any type of PC). To allow access to this expanded memory, you must have the appropriate 'DEVICE=..' statement in your CONFIG.SYS file; see the documentation accompanying your expanded memory board for the details on how to do this, or
- 2) You have a '386 or '486 PC with no expanded memory, but with additional memory (extended memory) above the 1Mb boundary. In such a case, you can use the EMM386.EXE device driver supplied with DOS to simulate expanded memory in extended memory. A typical CONFIG.SYS file for this configuration would contain the following lines:

```
DEVICE=C:\DOS\HIMEM.SYS
DEVICE=C:\DOS\EMM386.EXE
```

You can add additional parameters to the DEVICE command for EMM386.EXE to control the amount of expanded memory to be created (simulated) in your system.

If your computer does not fit either of the two categories above, but it does have extended memory (memory beyond the 1Mb boundary), you can use a file for virtual heap backing store. However, rather than using a file on your hard disk, create a RAM disk in extended memory and, using the '-vmfn' command line option, designate the backing store file as a file on that RAM disk drive. This is tremendously faster than using an actual hard disk file.

-vme <pages_per_allocation_request>

If you have Expanded Memory (LIM-compatible) in your computer, the Host Software can use it as a backing store medium for virtual heap memory. To enable access to Expanded Memory, you must have the appropriate 'DEVICE=...' statement in your 'CONFIG.SYS' file. See the documentation accompanying your Expanded Memory board for the specific details on how to do this.

The '-vme <pages_per_allocation_request>' option specifies the number of pages (16Kb per page) of Expanded Memory to request from the Expanded Memory Manager (EMM) each time the Host Software needs more Expanded Memory.

Whenever a program requests some number of logical pages of expanded memory from the EMM, those pages are internally associated with a 'handle'. The total number of available handles, as well as the total number of available logical pages of expanded memory, are resources which could easily be exhausted. If the total number of EMM handles in your system is small, make sure that the value of '-vme <pages_per_allocation_request>' is comparatively large. This will ensure that the available handles are not exhausted before the available logical pages of expanded memory.

To disable use of Expanded Memory as a backing store medium altogether, specify '-vme 0'.

-vmf <num_bufs> <buf_size>

Specifies the number and size of the real-memory buffers to be used to support file backing store as a medium for virtual memory. If either <num_bufs> or <buf_size> is zero, file backing store will not be used at all (i.e., it will be disabled). The integer specified for <buf_size> is the number of K-bytes to allocate for each real-memory buffer.

You can change the path/name of the default backing store file (\$VHEAP\$) using the '-vmfn' option.

To disable use of the file backing store medium altogether, specify '-vmf 0'.

-vmfn <file_name>

If your system has only Extended Memory, but no Expanded Memory, you can significantly speed up access to the backing store file by creating a RAM disk in Extended Memory and then specifying the backing store file name as a file on that RAM disk (e.g., '-vmf E:\TMPFILE', where E: is the RAM disk drive).

DEVICE=C:\DOS\HIMEM.E32
DEVICE=C:\DOS\EMM386.E32

Misc | Virtual Memory Command

The *Misc | Virtual Memory* command displays the following information about the current usage of virtual heap memory:

Expanded Memory Backing Store

EMM Version is the version number of the Expanded Memory Manager (EMM) device driver installed in your system. You can use EMM versions 3.2 and beyond.

Page Frame Segment is the segment address of the 64K area called the page frame. This will be somewhere above the conventional memory area used by DOS (0-640K) and below the 1M boundary of the upper memory area. The actual address depends on your computer's configuration.

Mappable Physical Pages is the number of physical pages in the upper memory area (640K-1M) of your computer into which expanded memory pages can be mapped (or swapped). The number here is the sum of the:

- 1) number of pages in the Page Frame (usually 4),
- 2) number of additional mappable pages above the 640K boundary.

Note that your system may have other mappable pages below the 640K boundary, but the Host Software will not use these pages.

This number represents the amount of expanded memory that the Host Software can reference directly at any particular time. The larger the value, the more likely that the memory area/node being referenced is directly accessible, and the less likely that the Host Software will need to swap pages to access the desired memory area/node.

Total Pages is the total number of expanded memory pages in your computer. Each page contains 16K of memory.

Pages Available is the number of pages currently available for allocation by the EMM to the Host Software. The remainder, if any, of the expanded memory pages in your system are unavailable for use by the Host Software because they are allocated to other programs or device drivers.

Pages Allocated to Host Software is the number of pages of expanded memory currently allocated to the Host Software.

Currently In Use by Host Software is the percentage of expanded memory which the Host Software is currently using from the pool of expanded memory pages allocated to the Host Software. When this percentage reaches 100%, the Host Software will request more expanded memory pages from the EMM; the '-vme' command line option (page B-2) specifies how many pages will be requested.

File Backing Store

Real-Memory Buffers is the number of conventional memory buffers being used to swap blocks in from and out to the backing store file. The '-vmf' command line option (page B-2) specifies how many buffers to use.

Buffer (Block) Size is the size of each conventional memory buffer used to swap blocks in from and out to the backing store file. The '-vmf' command line option (page B-2) specifies this buffer size.

The product of Real-Memory Buffers times Buffer (Block) Size represents the amount of real memory that the Host Software can reference directly at any particular time. The larger the value, the more likely that the memory area/node being referenced is directly accessible, and the less likely that the Host Software will need to swap blocks to/from the backing store file to access the desired memory area/node.

File Blocks Allocated is the number of blocks (of size Buffer (Block) Size each) currently needed by the Host Software. Essentially, this is the current size of the backing store file.

Currently In Use by Host Software is the percentage of file backing store which the Host Software is currently actually using. When this percentage reaches 100%, the Host Software will increase the size of the backing store file to create more virtual memory.

File size and number of blocks are used to describe memory as it is stored in memory buffer. The number of blocks needed to store the current amount of memory is calculated by dividing the current amount of memory by the size of a block. The number of blocks needed to store the current amount of memory is calculated by dividing the current amount of memory by the size of a block.

File size and number of blocks are used to describe memory as it is stored in memory buffer. The number of blocks needed to store the current amount of memory is calculated by dividing the current amount of memory by the size of a block. The number of blocks needed to store the current amount of memory is calculated by dividing the current amount of memory by the size of a block.

File size and number of blocks are used to describe memory as it is stored in memory buffer. The number of blocks needed to store the current amount of memory is calculated by dividing the current amount of memory by the size of a block. The number of blocks needed to store the current amount of memory is calculated by dividing the current amount of memory by the size of a block.

File size and number of blocks are used to describe memory as it is stored in memory buffer. The number of blocks needed to store the current amount of memory is calculated by dividing the current amount of memory by the size of a block. The number of blocks needed to store the current amount of memory is calculated by dividing the current amount of memory by the size of a block.

File size and number of blocks are used to describe memory as it is stored in memory buffer. The number of blocks needed to store the current amount of memory is calculated by dividing the current amount of memory by the size of a block. The number of blocks needed to store the current amount of memory is calculated by dividing the current amount of memory by the size of a block.

File size and number of blocks are used to describe memory as it is stored in memory buffer. The number of blocks needed to store the current amount of memory is calculated by dividing the current amount of memory by the size of a block. The number of blocks needed to store the current amount of memory is calculated by dividing the current amount of memory by the size of a block.

File size and number of blocks are used to describe memory as it is stored in memory buffer. The number of blocks needed to store the current amount of memory is calculated by dividing the current amount of memory by the size of a block. The number of blocks needed to store the current amount of memory is calculated by dividing the current amount of memory by the size of a block.

File size and number of blocks are used to describe memory as it is stored in memory buffer. The number of blocks needed to store the current amount of memory is calculated by dividing the current amount of memory by the size of a block. The number of blocks needed to store the current amount of memory is calculated by dividing the current amount of memory by the size of a block.

File size and number of blocks are used to describe memory as it is stored in memory buffer. The number of blocks needed to store the current amount of memory is calculated by dividing the current amount of memory by the size of a block. The number of blocks needed to store the current amount of memory is calculated by dividing the current amount of memory by the size of a block.

File size and number of blocks are used to describe memory as it is stored in memory buffer. The number of blocks needed to store the current amount of memory is calculated by dividing the current amount of memory by the size of a block. The number of blocks needed to store the current amount of memory is calculated by dividing the current amount of memory by the size of a block.

File size and number of blocks are used to describe memory as it is stored in memory buffer. The number of blocks needed to store the current amount of memory is calculated by dividing the current amount of memory by the size of a block. The number of blocks needed to store the current amount of memory is calculated by dividing the current amount of memory by the size of a block.

File size and number of blocks are used to describe memory as it is stored in memory buffer. The number of blocks needed to store the current amount of memory is calculated by dividing the current amount of memory by the size of a block. The number of blocks needed to store the current amount of memory is calculated by dividing the current amount of memory by the size of a block.

File size and number of blocks are used to describe memory as it is stored in memory buffer. The number of blocks needed to store the current amount of memory is calculated by dividing the current amount of memory by the size of a block. The number of blocks needed to store the current amount of memory is calculated by dividing the current amount of memory by the size of a block.

